Szabad Szoftver Konferencia és Kiállítás 2011

követő kiadvány

fsf.hu alapítvány

ISBN 978-963-89486-0-1

A konferencia támogatói

Főtámogató

Kiemelt támogató

Kiemelt médiatámogató

Támogatók

Médiatámogató

Szabad Szoftver Konferencia és Kiállítás 2011.

(követő kiadvány)

Budapest, 2011.

A kiadvány tördelése a TEX3.141592 verziójával készült,
a borítóhoz GIMP-et és LibreOffice-t használtunk.

A konferencia és a kiadvány szervezésében, elkészítésében a következők vettek részt:

Mátó Péter
: főszervező – stratégia, koordináció, web

Kiss Gabriella
: helyszín és catering szervezés, követőkiadvány korrektúrája

Muzslai Erzsébet
: előadók és nonprofit kiállítók szervezése

Baráth Gábor
: követőkiadvány borítója

Bőle György
: támogatók és kiállítók szervezése

Erdei Csaba
: stratégia, pénzügy

Kelemen Gábor
: követőkiadvány korrektúrája

Szántai István
: követőkiadvány korrektúrája

Tímár András
: követőkiadvány korrektúrája

Zelena Endre
: követőkiadvány tördelése

ISBN 978-963-89486-0-1
9 789638 948601 >

FSF.hu Alapítvány
URL: *http://www.fsf.hu/*
E-mail: *fsf@fsf.hu*

Jelen kiadvány a Creative Commons „Nevezd meg! – Ne add el! – Ne változtasd! 2.5” licenc alapján szabadon terjeszthető.

Előszó

2011-ben újra megrendezésre került a budapesti Szabad Szoftver Konferencia és Kiállítás.

A szervező FSF.hu Alapítvány legfontosabb célja a szabad szoftverek hazai népszerűsítése, ugyanakkor nagyon fontosnak tartjuk a hazai szabad szoftveres közösség együttműködését. Ennek szellemében minden olyan hazai civil szervezetet és céget meghívtunk kiállítónak, akik közreműködnek a szabad szoftverek és ahhoz köthető szellemiség építésében, népszerűsítésében.

A konferencián számos ismert és elismert előadó tartott színvonalas szakmai előadásokat. Ebben az évben a korábban tisztán szakmai program kiegészült egy egész napos közigazgatási szekcióval, ahol a kormányzat, önkormányzatok és sikeres nagy projektek képviselői adtak betekintést a szabad szoftverekkel kapcsolatos munkájukba, sikereikbe, stratégiájukba.

A konferencián több, mint négyszáz hallgató tekintette meg az előadásokat, melyekről videofelvétel is készült.

A videókat legegyszerűbben a konferencia http://konf.fsf.hu címen található weboldaláról lehet elérni.

Az előadók a bemutatóik mellé írtak egy-egy összefoglaló-, kiegészítő anyagot, ezeket az anyagokat gyűjtöttük össze ebbe a könyvbe. Reméljük, hasznos olvasmány lesz minden érdeklődő számára.

Tartalom

Ádám Szilveszter: Java –mi újság a licenc fronton? 7

Ancsin Gábor: GeoGebraMobile – a GeoGebra JavaScript verziója 15

Botyánszki Botond: Az nxlog naplózó rendszer 23

Czakó Krisztián: A rendszergazda barát Linux 32

Czanik Péter: syslog-ng web GUI-k 43

Dr. Dudás Ágnes: Haditudósítás a gigászok harcáról 53

Hargitai Zsolt: XenUI – Xen User Interface 63

Karay Tivadar: Áttörni a falat – szabad szoftverek egy önkormányzatban 67

Kis Gergely: Android – két évvel később 77

Kiszel Kristóf: A KDE jövője 91

Kövesdán Gábor: Heurisztikus regex mintaillesztés a FreeBSD-ben a TRE könyvtár segítségével 99

László Gábor: Nyílt kormányzat, „zárt ablakok”? 105

Németh László: LibreOffice – úton a DTP felé 113

Pfeiffer Szilárd: Zorp, a pokoli operátor tűzfala 123

Szegfű László: Nyílt forráskódú megoldások a közigazgatásban 129

Tímár András: LibreOffice 137

Torma László: Az Ubuntu és a Unity 145

Vajna Miklós: RTF támogatás a LibreOffice Writer programban 153

Vombert István: Többszálas (multi threaded) programozás Linux környezetben 161

Előadások

Java – mi újság a licenc fronton?

Ádám Szilveszer

Kivonat

A Java platform az utóbbi évtizedben az egyik legnépszerűbb szoftverfejlesztési és -futtatási környezetté vált, implementációi mára elérhetőek a legkülönfélébb hardverkörnyezetekhez és operációs rendszerekhez, a nagy teljesítményű szerverektől az intelligens kártyákig.

A Java platform egységének fenntartásában a mögötte álló nagyvállalatok piaci erején és elkötelezettségén túl mind a szerzői jog, mind pedig az iparjogvédelem körébe tartozó védjegyjog szabályainak komoly szerepük volt.

A mostani alkalommal arra vállalkozunk, hogy bemutassuk azokat a fontosabb jogi problémákat, amelyek a Java platform, kiemelten pedig a Java virtuális gép (JVM) és az alapvető futásidejű könyvtár (Java Standard Class Libraries) fejlesztői, valamint az ezeket más rendszerekkel integrálni szándékozók számára jelentkeznek. Ezt követően sorra vesszük a jelenleg (2011 őszén) jelentősebb felhasználói közösséggel rendelkező Java megvalósításokat és az ezekhez tartozó licencfeltételeket, rávilágítva azokra az akadályokra is, amelyek miatt jelentős számú, részben egymást is átfedő célkitűzésekkel rendelkező projekt dolgozott egy időben különféle alternatív Java megvalósításokon. Legvégül pedig röviden kitérünk arra is, hogy ezek a problémák hogyan nyernek jelentőséget a jelenleg legnépszerűbbnek számító okostelefon- és táblagépplatform, az Android körüli csatározásokban.

Tartalomjegyzék

1. Bevezetés **8**

2. Jelentősebb problémák az alternatív implementációk számára **8**

3. Leltárfelvétel – a legnépszerűbb Java implementációk **9**

3.1. Oracle JRE / JDK 10
3.2. OpenJDK 10
3.3. IcedTea 11
3.4. Apache Harmony 11
3.5. GNU Compiler for the Java programming language (gcj) 12
3.6. GNU Classpath projekt 13

4. Mi köze ennek az Androidhoz? **13**

1. Bevezetés

A Java platform az utóbbi évtizedben az egyik legnépszerűbb szoftverfejlesztési és -futtatási környezetté vált, implementációi mára elérhetőek a legkülönfélébb hardverkörnyezetekhez és operációs rendszerekhez, a nagy teljesítményű szerverektől az intelligens kártyákig. A Java programozási nyelv jelenleg is dobogós helyen szerepel a legnépszerűbb programozási nyelvek rangsorában. Ez a siker nem kis részben annak volt köszönhető, hogy a nyelv és a platform folyamatos fejlesztése közben sikerült kitartani az egyik legfontosabb eredeti célkitűzés mellett: az egyszer megírt szabványos Java kód bármilyen, a specifikációknak megfelelően működő Java futtató környezetben változtatás nélkül lefuttatható.

A Java platform egységének fenntartásában a mögötte álló nagyvállalatok piaci erején és elkötelezettségén túl mind a szerzői jog, mind pedig az iparjogvédelem körébe tartozó védjegyjog szabályainak komoly szerepük volt. Ugyanakkor éppen ezeknek az egységet fenntartani hivatott szabályoknak a merevsége az évek során sok vitának is forrása volt mind a Java platform megvalósításán dolgozók, mind a platform használói körében, mivel – más környezetekhez képest – kevésbé tették lehetővé a platform mindenkori hivatalos gazdájától független implementációk készítését és terjesztését. Amikor a Sun Microsystems Inc. 2006-ban bejelentette, hogy a Java platform általa készített referenciaimplementációját nyílt forrásúvá teszi, úgy tűnhetett sokak számára, hogy végre komoly elmozdulás történik ezen a területen is. Amikor pedig a Sun azt is bejelentette, hogy a Java platform következő nagy verzióváltása után már a fejlesztés alapját is egységesen a szabad szoftverként elérhető kódbázis fogja képezni, a korábbi viták végképpen lezárhatónak tűntek.

Ma, a nevezetes bejelentés után öt évvel azonban azt láthatjuk, hogy bár a Java platform és programozási nyelv népszerűsége töretlenül növekedett, az ide tartozó technológiákra vonatkozó szoftver licencelési feltételek és a kapcsolódó védjegyek használatával kapcsolatos előírások továbbra is megválaszolandó kérdéseket vetnek fel a mindennapi gyakorlatban. Sőt, ezek mellett mostanra az informatikai ipar egészéhez hasonlóan itt is egyre nagyobb a szerepe a szabadalmaknak is, amelyek – elsősorban a velük rendelkezők szándékainak kiszámíthatatlansága és az érvényesíthetőségükkel kapcsolatos bizonytalanságok miatt – a független megvalósításoknak további gátját jelenthetik.

A mostani alkalommal arra vállalkozunk, hogy bemutassuk azokat a fontosabb jogi problémákat, amelyek a Java platform, kiemelten pedig a Java virtuális gép (JVM) és az alapvető futásidejű könyvtár (Java Standard Class Libraries) fejlesztői, valamint az ezeket más rendszerekkel integrálni szándékozók számára jelentkeznek. Ezt követően sorra vesszük a jelenleg (2011 őszén) jelentősebb felhasználói közösséggel rendelkező Java megvalósításokat és az ezekhez tartozó licencfeltételeket, rávilágítva azokra az akadályokra is, amelyek miatt jelentős számú, részben egymást is átfedő célkitűzésekkel rendelkező projekt dolgozott egy időben különféle alternatív Java megvalósításokon. Legvégül pedig röviden kitérünk arra is, hogy ezek a problémák hogyan nyernek jelentőséget a jelenleg legnépszerűbbnek számító okostelefon- és táblagépplatform, az Android körüli csatározásokban.

2. Jelentősebb problémák az alternatív implementációk számára

A Java platform kifejlesztésekor a kezdeményező szerepet betöltő Sun Microsystems egy olyan ökoszisztémát akart létrehozni, amely egyrészről kellően nagy szabadságot ad a mellette döntő fejlesztőknek arra, hogy a Sun referenciaimplementációja mellett potenciálisan több szereplő által felkínált versenyző megoldásokból választhassanak, másrészről azonban meg akarta akadályozni az egységes Java platform feldarabolását egymással nem, vagy csak részben kompatibilis konkurens megoldások révén. Ez a törekvés érthető is volt az előző évtizedekben a kereskedelmi UNIX típusú rendszerek körében végbement fejlemények tükrében, ahol az egyes cégek által kínált UNIX variánsok közötti növekvő inkompatibilitás végül az egész platformot gyengítette meg, és ilyen módon egy külső ver-

senytárs, a Microsoft megerősödését segítette elő. Ezért, bár a Sun idővel egyre jobban megnyitotta a Java platform további fejlesztését tőle független szereplők előtt is, akik a Java Community Process révén az irányadó specifikációk kidolgozásának aktív részesei lehettek, de a különféle licencelési szabályok és a kezében lévő védjegyek használatának korlátozása segítségével mindig gondoskodott arról, hogy egy-egy implementáció megfelelőségéről maga mondhassa ki a végső szót. Ezen a téren a legjelentősebb akadályt az jelentette és jelenti a potenciális versenytársak előtt, hogy a "Java" megnevezést és a kapcsolódó védjegyeket csak azok az implementációk használhatják, amelyek sikerrel teljesítették az adott specifikációhoz tartozó Technology Compatibility Kit (TCK) tesztjeit. Ugyanakkor már a TCK-hoz való hozzáférés sem volt mindenki számára nyitott, így különösen a szabad szoftver projektek számára jelentett komoly problémát a megfelelőségi tesztelés. Mint látni fogjuk, azóta ezen a téren voltak változások, de még mindig nem beszélhetünk egyenlő hozzáférési lehetőségről minden projekt számára. A TCK tesztek sikeres teljesítése mellett további feltétel, hogy az érintett fejlesztő, vagy csoport megállapodjon a védjegyek használatáról a jogosulttal (Sun, majd Oracle). Mivel ilyen megállapodást nem mindenkinek sikerült tető alá hoznia, ma is nem egy projektet találunk, amelynek kínosan kerülnie kell még a "Java" szó használatát is a saját termékével kapcsolatban, ami viszont megnehezíti az alkalmazásfejlesztők közösségében a szükséges ismertség és bizalom megszerzését.

A forráskód megnyitása előtt csak nagyon kevés valóban használható alternatív megvalósítás létezett, ezek fejlesztésének nehézsége miatt a gyakorlatban a legtöbb esetben a Sun által készített referenciaimplementációkat (Java Runtime Environment – JRE, Java Development Kit – JDK) használták. Ezek – bár a legtöbb esetben ingyenesen beszerezhetők voltak – zárt forrású kereskedelmi termékek voltak, amelyeknek a terjesztését is csak meghatározott feltételek teljesítése mellett engedélyezte a Sun. További problémát jelentett, hogy a Sun a JRE-t és a JDK-t is csak a saját megítélése szerint jelentősebb hardverarchitektúrákra és operációsrendszer-környezetekre készítette el, így például a Mac OS és a Linux hosszú ideig nem rendelkezett támogatással. Ezért a Java platform történetében valóban nagy jelentősége volt a Sun azon döntésének, hogy a Java környezet alapvető alkotóelemeit, így mindenekelőtt a JVM-et és az alapvető futásidejű könyvtárat nyílt forrásúvá teszi, kivéve azokat a komponenseket, amelyek harmadik féltől származó kódot tartalmaztak. Középtávon azonban itt is az volt a cél, hogy ezeket szintén megnyissák, vagy azonos funkcionalitású szabad szoftveres licencű komponensekkel helyettesítsék. A bejelentést követő fél év alatt a továbbra is zártan maradó komponensek arányát 5% alá csökkentették mind az aktuálisan termékként a piacon elérhető, mind pedig a következő jelentős verzióugrást jelentő 1.6-os verzió alapját adó kódbázis esetében, a későbbiekben pedig sikerült ezeket lényegében eliminálni. (A zárt licencű komponenseket a Sun továbbra is használta a saját, csak bináris formában terjesztett JRE és JDK kiadásaihoz) A Java platform soron következő verziójának (Java 7) fejlesztése pedig már eleve legnagyobb részben a forráskód megnyitása után zajlott, így ott az ezzel kapcsolatos követelményeket is időben figyelembe lehetett venni. Ettől függetlenül az alternatív implementációk fejlesztése nem lett feltétlenül könnyebb, mint azt a következőkben látni is fogjuk.

3. Leltárfelvétel – a legnépszerűbb Java implementációk

Ebben a fejezetben sorra vesszük a jelenleg használt legelterjedtebb Java megvalósításokat a rájuk vonatkozó licencfeltételekkel együtt, ideértve a zárt forrású programokat és a szabad szoftveres projekteket is.

3.1. Oracle JRE / JDK

A forráskód 2006-2007-ben végbement megnyitását követően a Sun, majd a céget és vele együtt a Java platformhoz kötődő technológiákat és jogosultságokat is felvásárló Oracle Corporation is folytatta a bináris formában elérhetővé tett, zárt forrású referenciaimplementáció megjelentetését. Jelenleg a JDK és a JRE Solaris, Windows és Linux környezetekre érhető el 32 és 64 bites x86 architektúrákra. Az ingyenesen, az Oracle honlapjáról letölthető alapszintű JDK mellett a cég kínál további, többletfunkciókat is tartalmazó verziókat is díjfizetés ellenében (Java SE Advanced, Java SE Suite). A JVM és az alapvető futásidejű könyvtár mellett mindhárom verzió tartalmazza a Java appletek megjelenítéséhez szükséges böngészőbővítményt, valamint a weben elérhető Java programok előzetes telepítés nélküli futtatását lehetővé tevő Java Web Start (JWS) összetevőt is. Noha a később bemutatandó nyílt forrású OpenJDK projekten alapulnak, mind futtatásuk, mind pedig a szoftver beszerzése, példányainak többszörözése és terjesztése csak korlátozottan, az "Oracle Binary Code License Agreement" által meghatározott körben és feltételekkel megengedett. Ez a gyakorlatban azt jelenti, hogy amíg az egyes felhasználók saját céljaikra letölthetik, többszörözhetik, telepíthetik és használhatják ezeket, a terjesztés változatlan formában is már csak akkor megengedett, ha az egy olyan saját program mellé csomagoltan történik, amelynek futtatásához a JRE (és esetleg a JDK Oracle által külön meghatározott részei) szükséges, és amely jelentős többletértéket jelentő funkciókat tartalmaz a JRE-hez képest. Külön is tiltott a terjesztés bármilyen olyan más szoftverrel együtt, amelynek célja az Oracle által biztosított JRE bármely részének lecserélése, illetve kiváltása.

Ez a gyakorlatban azt jelenti, hogy ezt a JRE-t/JDK-t például fő szabály szerint nem lehet egy operációs rendszer környezettel együtt terjeszteni, mert ott a fentebb írt feltételek nem teljesülnek. (Érdekes megjegyezni, hogy kimondottan a különféle Linux terjesztésekbe való integrálás lehetővé tétele érdekében korábban létezett egy "Operating System Distributor License for Java" is, de ezt az Oracle 2011. augusztus 29-cel visszavonta.)

3.2. OpenJDK

Az OpenJDK projekt azoknak a forráskódelemeknek a bázisán jött létre, amelyeket a Sun 2006-tól kezdve adott közre szabad szoftverként. Legelőször a HotSpot nevű virtuális gép került kiadásra, majd 2007-ben a teljes alapvető futásidejű könyvtár (Java Standard Class Library), kivéve azokat az elemeket (így például a grafikus felület egy részét), amelyeket a Sun-nak harmadik személyek fennálló jogai miatt nem volt lehetősége nyílt forrásúvá tenni. Ezeknek aránya kezdetben 4% volt a teljes könyvtárkódból, később azonban a hiányzó részek mindegyikét vagy kiadták a szükséges engedélyek beszerzése után, vagy pótolták más, azonos funkciójú, de szabadon kiadható részekkel. A JDK részét képező parancssoros eszközök, pl. a javac nevű fordítóprogram szintén szabad szoftverként kerültek közreadásra. Nem adták ki viszont a böngészőbővítményt és a Java Web Start programot sem, noha a Sun ezt többször is megígérte az idők folyamán. Ma az OpenJDK projekt két fő ággal rendelkezik, az egyik (OpenJDK6) a Java platform 1. 6-os verzióját követi, a másik pedig (OpenJDK) a 7.0-s verziót. A korábban írtaknak megfelelően a Java 7.0 referenciaimplementációja már egyértelműen az OpenJDK alapján készült. (A Java 1.6 esetében a helyzet annyiban volt más, hogy ott a forráskód publikálása már az első általános felhasználásra szánt kiadás elkészítése utáni időre esett, így az OpenJDK6 projekt kódbázisa nem teljesen azonos a Sun, majd később az Oracle által továbbfejlesztett 1.6 referenciamegvalósítás egyes verzióihoz publikált forráskódcsomagokkal, amelyeknek a licencfeltételei is mások.)

A Sun a szabad szoftverként kiadott JDK és JRE licencelésére a GNU General Public License második verzióját (GPLv2) választotta az úgynevezett kapcsolási kivétellel (linking exception). Ez azt jelenti, hogy – ellentétben a GPL általános szabályával, amely szerint bármely, a GPL alatt kiadott program, vagy programrész bármely módosított változata, vagy azon alapuló származékos mű éppen úgy a GPL hatálya alá kerül, mint bármely olyan mű is, amivel a GPL alatt álló programot,

vagy programrészt (például egy könyvtárat) dinamikusan, vagy statikusan kapcsolják (dynamic, illetve static linking) – itt a GPL alatt álló művet bármilyen más nyílt, vagy nem nyílt forrású licenc alatt kiadott más programhoz lehet kapcsolni, feltéve, hogy az új program nem a GPL alatt álló program, vagy programrész módosításával keletkezett származékos mű. Ezzel a megoldással kívánta a Sun biztosítani, hogy az általa kiadott JDK és JRE forrásokból készített binárisokkal továbbra is lehessen nem szabad szoftverként kiadott Java kódot is fordítani, értelmezni és futtatni. (Hasonló megoldást alkalmazott a Free Software Foundation is a GNU Compiler Collection – GCC esetében, hogy lehetővé tegyék a GCC felhasználását nem szabad szoftverek fordítására is.)

Fontos kiemelni, hogy sem az eredeti, Sun által kiadott forráskódot, sem pedig az OpenJDK projektek kódját nem lehet pusztán a szokásos, fejlesztői munkaállomásokon általában meglévő eszközökkel lefordítani, hanem ehhez további binárisokra van szükség, ezeket külön tették elérhetővé más licencfeltételek alapján. Emellett az OpenJDK projekt keretében csak forráskód közzétételére kerül sor, kész, bináris formában elérhető JDK, illetve JRE nem tölthető le. Ehelyett a felhasználókra vár a feladat, hogy a forráskód, a külön letöltött binárisok és a közzétett utasítások alapján a binárisokat létrehozzák. Mivel ezeket így nem a Sun, illetve az Oracle hozza létre, ezért nem is mennek át tesztelésen abból a szempontból, hogy megfelelnek-e az aktuális specifikációknak. Viszont az eredmény csak akkor tekinthető Java kompatibilisnek, ha ezek a tesztelések eredményesen lezajlottak. Annak érdekében, hogy az OpenJDK felhasználóknak megkönnyítsék a megfelelőség tesztelését, a Sun, majd az Oracle számukra hozzáférhetővé tette a Technology Compatibility Kitet (TCK) egy speciális licenc alapján, amely kimondottan csak az OpenJDK és az annak felhasználásával készült projektek megfelelőségének ellenőrzésére engedélyezi a használatot. (OpenJDK Community TCK License Agreement – OCTLA). Természetesen ha valaki a Java nevet, logót, vagy más, kapcsolódó védjegyet is használni kívánja az általa készített binárisokra, akkor arra külön kell védjegyhasználati megállapodást kötnie az Oracle Inc-vel.

Az OpenJDK projekt a fentebb jelzett specialitásai ellenére is gyorsan népszerű lett, a különféle Linux terjesztések számára készült kész csomagok (gyakran a hivatalos, Sun, illetve Oracle által kiadott JDK/JRE mellett), valamint a Windows és a Linux mellett több más operációs rendszer platformon is működő változatok állnak rendelkezésre (így például FreeBSD-re, vagy MacOS X-re.)

3.3. IcedTea

Az IcedTea projektet az az igény hívta életre, hogy az OpenJDK projekt keretében kiadott forráskódot olyan Linux terjesztéseknél is fel lehessen használni csomagok készítésére, amelyek nem engedik a zárt forrású eszközök használatát erre a célra. (Ilyen például a Fedora projekt.) Emellett céljai között szerepelt az OpenJDK projektben hiányzó böngészőbővítmény és Java Web Start komponensek nyílt forrású helyettesítőinek kifejlesztése is (IcedTea-Web projekt). Elnevezését azért kapta, mert a Java védjegy használatára nem rendelkeztek engedéllyel a fejlesztők. Mivel a céljai között szerepel, hogy az itt már kipróbált és megfelelőnek bizonyult változtatások idővel bekerüljenek az OpenJDK forrásfájába is, ezért azzal azonos licencelési szabályokat használnak (GPLv2 + kapcsolási kivétel). Az IcedTea projekt eredményeképpen lehetségessé vált az, hogy az OpenJDK forrásokat kizárólagosan szabad szoftverek felhasználásával lefordíthassa egy fejlesztő. Emellett jelenleg csak ennek a projektnek a keretében érhető el olyan böngészőbővítmény, amely natívan támogatja a 64 bites böngészőket.

3.4. Apache Harmony

A Harmony projekt célja egy, a Sun-tól független, nyílt forrású Java implementáció létrehozása, kezdetei pedig még régebbre nyúlnak vissza, mint az OpenJDK fejlesztéseké. Eredetileg a Java platform

1.5-ös verziójának megfelelő JDK közreadása volt a cél, de később megkezdték az 1.6-os verzió specifikációjának megvalósítását is. A projekt licencelési okokból először teljesen nulláról kezdett, de szívesen fogadott hozzájárulásokat egyéni fejlesztőktől és cégektől is, így az első években viszonylag gyorsan fejlődött. Mint az Apache Software Foundation hivatalos projektjei, a Harmony is az Apache License használata mellett döntött. Ez egy megengedő típusú szabadszoftverlicenc, mivel alapvetően nem korlátozza, hogy az alá tartozó programokat vagy programkomponenseket milyen célra lehet használni, és nem állít korlátokat a módosítások, valamint a módosított változatok terjesztése elé sem. Csupán azt kívánja meg, hogy a módosítások legyenek megjelölve, valamint az eredeti szerzőkkel és szerzői jogokkal kapcsolatos jelölések maradjanak épségben. Emellett az Apache License 2.0-ás verziója már előírja azt is, hogy amennyiben egy módosítás használatához szükségszerűen olyan szabadalmak használata is szükséges, amelyeknek a módosítás szerzője a jogosultja, akkor a módosított verzió használója automatikusan engedélyt kap ezeknek a szabadalmaknak a használatára is.

Bár a Harmony projektnek hangsúlyozottan célja volt az is, hogy a különböző Java implementációkon dolgozó fejlesztők és fejlesztői csoportok együttműködését és közös munkáját elősegítse, a licencfeltételek különbözősége ezt a gyakorlatban eléggé megnehezítette. Mivel az Apache License és a GPLv2 egymással nem összeegyeztethető szabályokat tartalmaz, (a GPLv2 semmilyen olyan licenccel nem kompatibilis, amelyik hozzá képest többlet korlátozásokat ír elő) a Harmony projekt nem használhatta fel a többi, ebben a fejezetben bemutatott szabadszoftver-kezdeményezés keretében írt és szabadon elérhető kódot, és fordítva sem volt ez lehetséges. A GPL 2007-ben elfogadott legújabb, harmadik verziója (GPLv3) az Apache License 2.0-s verzióját már kompatibilisnek ismeri el, de ez csak azokra a programokra, vagy programkomponensekre hathat ki, amelyeket már eleve ilyen licenc alatt adtak ki, vagy legalább a feltételek lehetővé teszik ugyanazon licenc későbbi verziójának használatát is. (Sok GPLv2 alatt kiadott program esetén ez a lehetőség adott.) További problémát jelentett, hogy a Harmony projektnek nem sikerült a TCK használatára olyan feltételek mellett engedélyt kapnia, amely az Apache License feltételeinek megfelelően minden fejlesztő előtt nyitva állt volna, közvetlen felhasználási célra vonatkozó megkötés nélkül. Ezért az általuk készített JDK és JRE binárisok Java specifikációknak való megfelelőségét nem lehetett az előírt tesztek teljesítésével bizonyítani, ami az OpenJDK-val és a rá épülő projektekkel szemben jelentős hátrányt jelentett. Mára a Harmony projekt több jelentős támogató távozását követően gyakorlatilag felfüggesztődött.

3.5. GNU Compiler for the Java programming language (gcj)

A Free Software Foundation (FSF) égisze alatt működő gcj projekt egy olyan fordítóprogramot fejlesztett ki, amely képes a Java forráskódot mind a szokásos JVM-ek által végrehajtható bináris kóddá (class fájlok) mind pedig a CPU által közvetlenül végrehajtható natív alkalmazáskóddá átalakítani. Része a különféle programnyelvekhez fordítóprogramokat tartalmazó GCC-nek (GNU Compiler Collection). A gcj önmagában nem volt alkalmazható, mivel a programok lefordításához egyéb komponensek (mindenekelőtt az alapvető futásidejű könyvtár – Java Standard Class Library) is szükségesek voltak. Ezért a gcj projekt szorosan együttműködött a később bemutatandó GNU Classpath fejlesztőivel, azt is célul tűzték ki, hogy a libgcj nevű futásidejű könyvtárat összevezetik a másik megvalósítással a gyorsabb előrehaladás érdekében. A gcj – a GCC többi tagjához hasonlóan – a GPLv2, illetve GPLv3 licenc alatt érhető el, a kapcsolási kivétel itt is szerepel annak érdekében, hogy nem szabad szoftvernek minősülő programkód lefordításához is lehessen használni. Bár a gcj jelenleg is része minden GCC kiadásnak, jelentős fejlesztésekre az utóbbi időben már nem került sor.

3.6. GNU Classpath projekt

Az ugyancsak az FSF támogatásával létrejött GNU Classpath projekt azt tűzte ki célul, hogy egy szabad szoftverként elérhető alapvető futásidejű könyvtárat fog kifejleszteni, amely a Sun által kiadott referenciaimplementáció helyett is alkalmazható lesz. A gyakorlatban több, szabad szoftverként elérhető JVM- megvalósítás is a GNU Classpath kódját használja a saját futásidejű könyvtárának fejlesztéséhez, vagy teljesen át is veszi azt. Különösen szoros együttműködés alakult ki az előbb bemutatott gcj projekttel, de olyan, ma már kevésbé széles körben használt szabad JVM projektek is a használók között vannak, mint a SableVM vagy a Kaffe. Az IcedTea projekt is alkalmazott innen származó programrészeket a Sun által kiadott alapvető futásidejű könyvtár hiányzó moduljainak helyén addig, amíg azok kiváltása, vagy megnyitása meg nem történt. A projekt keretében jelenleg már inkább csak hibajavítási tevékenység zajlik. A GNU Classpath projekt is a gcj-vel azonos licenc alatt érhető el.

4. Mi köze ennek az Androidhoz?

Az Android egy elsősorban hordozható eszközökön használt operációs rendszer, amelyet a Google által vezetett Open Handset Alliance fejleszt, miután az eredeti fejlesztőt (Android Inc.) a Google 2005-ben felvásárolta. Az Android rendszerben a felhasználói programok egy virtuális gépben futnak (Dalvik virtual machine), amely elszigeteli őket egymástól és az operációs rendszertől. A fejlesztők által a Java programozási nyelv egy dialektusában írt programkódot a készülékre telepítés előtt átalakítják előbb bináris Java kóddá (class fájlok) , majd ebből hozzák létre a Dalvik Executable (.dex) fájlokat, amelyeket a Dalvik virtuális gép értelmezni tud. Bár a Dalvik nem tekinthető teljes JVM megvalósításnak, az alapvető futásidejű könyvtár egy jelentős részét implementálja, ehhez az Apache Harmony projekt által közzétett kódot is felhasználja. A Dalvik – az Android rendszer 2.3.x és előző verzióiban legalábbis – az Apache License alatt nyílt forrású szoftverként érhető el. Az ennél újabb verziók nyilvánosságra hozatalára a Google ígéretet tett ugyan, de erre eddig nem került sor.

Az elmúlt évben az Oracle az Egyesült Államokban pert indított a Google ellen, mivel állítása szerint a Dalvik implementálása során a Google megsértette a Sun szerzői jogait, illetve több olyan szabadalmat is jogosulatlanul használt fel, amelyeknek a Sun volt a jogosultja. A Google tagadta a felhozott állításokat, kiemelve, hogy a Dalvik fejlesztése során nem használta fel a Sun által hozzáférhetővé tett forráskódot, hanem saját, új megvalósítást készített, így nem kellett figyelembe vennie a Sun által publikált forráskóddal kapcsolatban fennálló korlátozásokat. Az Oracle állítása szerint ugyanakkor az Android fejlesztése során a felek tárgyalásokat is folytattak a kérdéses szabadalmak és szerzői jogilag védett programrészek használatáról, amelyek azonban akkor eredménytelenül zárultak, így azok létezéséről és mibenlétéről a Google-nek tudomása volt.

Az Oracle által állított szerzői jogi jogsértéssel kapcsolatban konkrétumok még nem ismertek jelenleg, ezért a konkrét ügyről nehéz véleményt mondani. Ugyanakkor tény, hogy amennyiben valamely fejlesztő olyan programrészeket használ fel, amelyeket a Sun, vagy az Oracle a GPL alatt tett közzé, vagy ilyen programrészeket felhasználva hozott létre származékos műveket, akkor ezekkel kapcsolatban köteles betartani a GPL terjesztéssel kapcsolatos előírásait, amelyek megkövetelik, hogy a binárisok mellett az azokhoz tartozó forráskódot is át kell adni, illetve elérhetővé kell tenni. Emellett figyelmet érdemel a GPLv3 azon előírása is, mely szerint amennyiben a GPL alatt nyilvánosságra hozott kódot valamely berendezésen telepítve terjesztik, akkor a forráskód mellett át kell adni azokat az információkat, illetve eszközöket is, amelyek ahhoz szükségesek, hogy egy módosítást követően a programot a berendezésen ismételten telepíteni lehessen. Itt pedig nem csak a Google-nek az a döntése vet fel kérdéseket, amely szerint az Android 3.0-val kezdődő verzióinak teljes forráskódját egy általa meghatározott későbbi időpontig nem hozza nyilvánosságra, hanem az

Androidot használó okostelefonok és táblagépek gyártóinak az a gyakorlata is, hogy az eszközön műszaki intézkedésekkel megakadályozzák tőlük független forrásból származó programok, mindenekelőtt operációs rendszer telepítését. Ez azonban már nem csak a Java platformmal kapcsolatos ügy, hiszen az Android fejlesztése során számos más nyílt forrású szoftverprojekt által készített programot, illetve programrészt használtak fel.

Mint az ilyen természetű eljárásokban általában, itt is várható, hogy a felek a köztük fennálló vitát idővel bíróságon kívül, egyezséggel fogják lezárni, azonban ennek időpontja még nem látható előre. Így valószínűleg hosszabb eljárásra lehet felkészülni, amely bizonytalanságot okozhat az Androidot, illetve Androidra fejlesztők körében, ugyanakkor újabb adalékot szolgáltathat a szabad szoftver licencek értelmezésével kapcsolatos gyakorlathoz.

GeoGebraMobile – a GeoGebra JavaScript verziója

Ancsin Gábor

Tartalomjegyzék

1. GeoGebra **16**
1.1. A GeoGebra hazánkban 16
1.2. A GeoGebra felépítése 17

2. GeoGebraMobile **18**
2.1. A probléma 18
2.2. GWT – Javából JavaScriptet 19

3. GeogebraWeb **21**
3.1. Git vagy Svn 21
3.2. Fejlesztés menete – meglett végül is 22

4. Konklúzió **22**

1. GeoGebra

A GeoGebra[1] egy interaktív matematikai program, mely összefűzi a geometriát, algebrát, táblázatokat, gráfokat, statisztikát és kalkulust egy könnyedén használható és felhasználóbarát programban. Teljesen dinamikus, interaktív, az általános iskoláktól az egyetemekig mindenhol használható. Felhasználóbázisa hatalmas, több mint 50 nyelvre létezik fordítás, valamint elérhetőek szerkesztések az interneten a http://www.geogebratube.org oldalon, a GeoGebrával készített dinamikus példák olyan adatbázisában, melyet a közösség tart fenn. Ez az adatbázis kereshető, szerkeszthető, azonnal kipróbálható példák sokaságát tartalmazza.

A szoftvert 2001-ben kezdte el fejleszteni szakdolgozati munkájaként Markus Hohenwarter, a Salzburgi Egyetem matematika szakos hallgatója (immáron a Linzi Egyetem matematikai didaktikai tanszékének vezető professzora). A Javában írt, platformfüggetlen, GNU GPL 3 alapú oktatási szoftver 2002 óta számos díjat, elismerést nyert (többek között European Academic Software Award 2002, Lernie Award 2005, AECT Award 2008, The Tech Awards 2009).

A szoftvert jelenleg 200 országban használják. Havonta 300 000 letöltés és 750 000 különböző látogató jellemzi a program elterjedtségét. 2008 óta a megrendezésre került 29 nemzetközi GeoGebra konferencián a világ több mint 50 országából számos fejlesztő, oktató és kutató szakember találkozott. A Nemzetközi GeoGebra Intézet égisze alatt 57 országban 75 GeoGebra intézet ad regionális támogatást a felhasználók számára, akiket már 30 megjelent kiadvány is segít, valamint 15 000 online elérhető tanítási segédlet és példa áll rendelkezésükre.

1.1. A GeoGebra hazánkban

Mielőtt a szakmai részbe keményen belevágnánk, pár szóban hadd említsem meg, hogy mi magyarok hol helyezkedünk el a GeoGebra világában. Ezúton szeretnék köszönetet mondani Papp-Varga Zsuzsannának, aki a magyar közösség egyik vezéralakja, és az itt következő információkat rendelkezésemre bocsájtotta.

Ha a http://www.geogebra.org/ látogatottsági statisztikáira pillantunk, láthatjuk, hogy Magyarországról kb. 4600 fős látogatottsággal a 28. helyen vagyunk (legelöl Franciaország 184 000 fővel). Az egyedi látogatók száma havonta több száztól több ezerig terjedhet, iskolai szünetekben természetesen kis visszaesés érzékelhető. A magyarországi fő tevékenységek: előadások, workshopok, fordítási munkálatok, magyar nyelvű segédanyagok készítése, módszertani kutatások, pályázatok, és nem utolsósorban a szoftverfejlesztés, mellyel ezen értekezés fő része fog foglalkozni.

A GeoGebra oktatása, matematika tanárok felkészítése előadások és workshopok formájában az egyetemeken tanárképzés keretében (ELTE, SZTE, EKF, NYF), konferenciákon és továbbképzések alkalmával is történik hazánkban. Diplomamunkák, segédanyagok születnek folyamatosan, melynek nagy része elérhető a magyar wikin[2].

Kiemelten említeném az FSF.hu Alapítvány által kiírt pályázatokat, melyek segítségével a szoftverfejlesztés és a közösség működése is szponzorálásra került és kerül. Az „Egy GeoGebrás Órám” pályázat, valamint a szoftverfejlesztés is sokat köszönhet az FSF.hu Alapítványtól kapott pályázati támogatásnak.

A kicsiny országunk polgárai a közösségi oldalakon (Facebook, iwiw, Google csoportok és a GeoGebra fórum) is aktív GeoGebra résztvevők. GeoGebra intézetek alakultak Miskolcon és a Kaposvárott, folyamatban van Pécsett, Szegeden és Budapesten. Az intézetekről (nemzetközi szinten) itt találhatunk információt: `http://www.geogebra.org/cms/institutes`. Ez egyben rálátást enged, mennyire világméretű is a projekt. A `http://www.geogebra.org/cms/team` oldalon pedig bemutatjuk, kik is vagyunk mi pontosan.

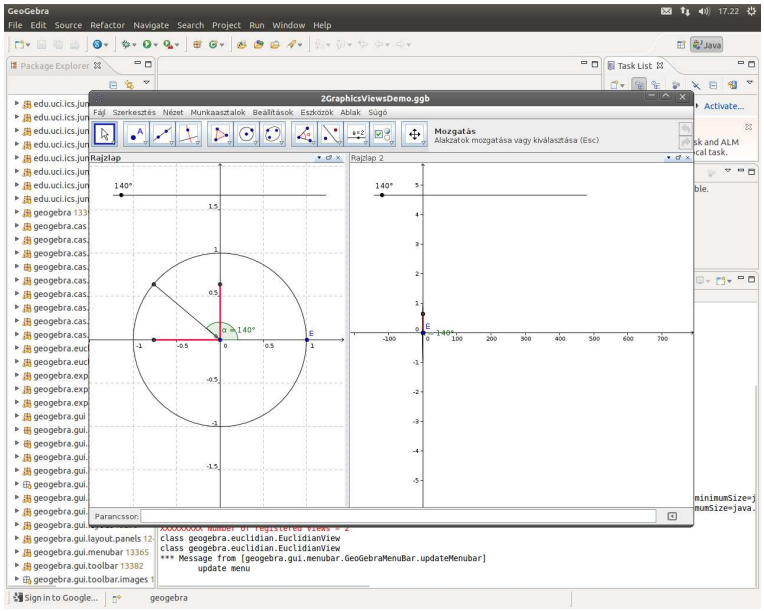
[1] `http://www.geogebra.org/`

[2] `http://www.geogebra.org/en/wiki/index.php/Hungarian`

1.2. A GeoGebra felépítése

Kedves olvasó, ha idáig eljutottál, vége a száraz bevezetőnek, kezdődhet az izgalmas szakmai rész! Ha a Java, Ant, autobuild, Eclipse, HTML5, JavaScript, CSS, GWT, Git, Svn szavak vagy ezek bármilyen kombinációja számodra „zavart kelt az Erőben”, javaslom, helyezkedj el kényelmesen. Mivel most egy több mint tíz éve folyamatosan fejlődő projektről fogok beszélni, amely igen szerteágazó, néhány dolgot csak részlegesen érintek, másokról többet beszélek. Bármilyen kérdés merülne fel benned, ne hezitálj emailt dobni a gabor@geogebra.org címre.

A GeoGebra egy Java nyelven íródott projekt, fejlesztése Eclipse-ben zajlik. (Én megszállott Vim rajongó vagyok. Egyszer hackeltem Javát Vim-ben... nem akarok beszélni róla). Két plugin, a JavaCC parser, valamint a Subversive Svn verziókezelő szükséges az alap Eclipse-en kívül. A megjelenítés Swing GUI-val történik. Hogy gyors elképzelésünk legyen a kinézetről, íme egy kép:



A program kezdetektől fogva működik applet verzióval is, mely HTML oldalakba ágyazott Java appletekkel éri el (szinte) azt a funkcionalitást, melyet a desktop verzió. Ez nagyon kézenfekvő módja a szerkesztések bemutatásának, megosztásának, főleg mióta a geogebratube.org élesben fut, mely gyakorlatilag csak erre a funkcióra hagyatkozik, és szervesen össze van kötve a desktop verzióval (azaz van Share gombunk, mellyel egy kattintással megoszthatunk a geogebratube-ra). És itt jön a nagy bumm, de ne rohanjunk előre.

A fejlesztés Svn (Subversion) verziókövetővel zajlik. A kb. 30 fejlesztő a világ minden tájáról egy SVN-repositoryba[3] küldi fel a fejlesztett kódot. A kód menedzselése, ticketek kezelése, különféle dokumentációk fenntartása trac[4] rendszerrel működik. Ez mind szép és jó, megszokott, összeállt rendszer. Tökéletes. Szeretjük. És ekkor...

[3] `http://www.geogebra.org/svn/trunk/geogebra/`
[4] `http://www.geogebra.org/trac`

2. GeoGebraMobile

„Apple hates Flash”. Arról sajnos már említés sem esik, hogy „Apple hates Java, too”. Pedig bizony. Hogy konkrétak legyünk, az esélye annak, hogy a GeoGebra (vagy egyéb Java – Swing alkalmazás) futni fog mobil (iPad, iPhone, Android stb.) eszközökön, erősen konvergál a zéróhoz. Ez pedig elég kellemetlen olyan program esetében, melynek lényege épp a webes megoszthatóság.

Szerencsére Markusnak 2009 szeptemberében eszébe ötlött, hogy jó lenne valamit kezdeni ezzel a ténnyel. Ebben az időben kezdődött el az Android telefonok elterjedése, és az iPad 2010 áprilisában nyitott. Így hát jó barátom és munkatársam, Kovács Zoltán továbbította nekem Markus levelét, melyben több projekt között támogatást nyert a „GeoGebra JavaScript Port”, mely később a GeoGebraMobile nevet kapta. Jómagam akkor már több éve foglalkoztam JavaScript programozással, és rábólintottam, hogy vállalom.

2.1. A probléma

Az szerettük volna elérni, hogy a .ggb kiterjesztésű fájlokat valamilyen úton-módon elérhetővé, kipróbálhatóvá tegyük mobil eszközökön is. A .ggb fájlok gyarkolatilag zip fájlok, egy xml-t tartalmaznak, mely maga a szerkesztés parse-olható leírása, valamint egy képet a szerkesztésről. Két lehetőség közül választhattunk:

1. Egy létező rendszer használata. Nem meglepő, hogy van olyan dinamikus matematikai rendszer, mely a böngészőket célozta meg platformként: ez a JSXGraph[5]. Egy JavaScriptben írt rendszer, mely egyéb matematikai szoftverek között a GeoGebra fájlokat is képes kezelni és megjeleníteni. A <canvas> és <svg> elemeket használja (böngészőtől függően), és egy egyszerű API-t ad, mellyel dolgozhatunk: gyakorlatilag matematikai szkriptnyelvet kapunk általa. Mindenkinek ajánlom, aki bármilyen vizuális 2D-s alakzatos képmanipulálós webes feladathoz nem akar nulláról indulni. Tehát ezzel a rendszerrel történt pár próbálkozás az első hónapokban, mely rávilágított arra, hogy mégsem jó ötlet. Ugyanis egy létező rendszer esetén a következő problémákkal kell szembenéznünk:
   - Különbözik mind arculatában, mind használatában, és egyes funkciók támogatása is hiányzik.
   - Mások a rajzoló algoritmusok, ezért a matematikai megjelenítés sem lesz egyforma.
   - Még ha a megjelenítés és a funkcionalitás ugyanaz is, a matematikai elemek dinamikus viselkedése, együttműködése különbözik – például a metszéspontok megjelenítése más.
2. A GeoGebra JavaScriptre való átírása (portolása). Mivel adottak a lehetőségek (HTML5 technológiák), mi jut eszébe egy ízig-vérig programozónak: oké, nézzük, mi a funkcionalitás, esetleg kukkantsunk a kódba, és írjuk meg JavaScriptben. Ez vonzó is, hisz így tényleg úgy optimalizáljuk, ahogy akarjuk, és imádunk JavaScriptben kódolni! Ám pillantsunk csak a terminálba picit:

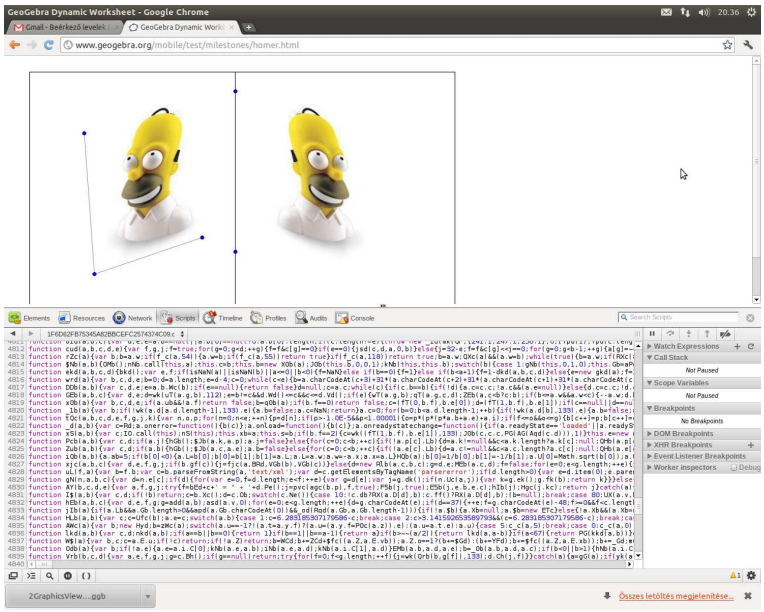
   ```
   apa@apa:~/M/workspaces/GeoGebra$ find -name *.java | wc -l
   3710
   apa@apa:~/M/workspaces/GeoGebra$ find -name *.java | xargs cat | wc -l
   921524
   ```

   Mint látjuk, 3710 java fájlról és összesen 921 524 sornyi kódról van szó. Ahogy említettem, ez egy több mint tíz éve fejlődő alkalmazás. Kézzel átírni bizony nem lenne egyszerű feladat. Marad hát a portolás! Ezt választottunk végül is.

[5] http://jsxgraph.uni-bayreuth.de/wp/

2.2. GWT – Javából JavaScriptet

Szerencsére léteznek megoldások Javából JavaScriptre való portolásra, fordításra. A leghasználhatóbbnak a GWT[6]-t találtuk. Több szempontból is megnyerő: Eclipse-ben történik a fejlesztés, a GWT plugin telepítésével. A Java JRE-nek elég nagy része támogatott, és mivel felhasználóbázisa hatalmas, rengeteg .jar könyvtár készül hozzá. Folyamatos fejlesztés alatt van, havonta jönnek ki frissítések (HTML5 támogatottság, touch események, a JRE kompatibilitás bővítése, böngészőoptimalizálás, hogy csak párat említsek). A fejlesztő által készített Java kódot több JavaScript fájlra (permutációra) fordítja le , melynek betöltése egy szelektor script segítségével történik, aszinkron módon. Így a különféle böngészők különféle .js fájlokat kapnak az optimális működés elérésének érdekében. Mivel a .js fájlok nem kerülnek „emberi fogyasztásra” – csak a böngészőnek kell értelmeznie őket – , teljes mértékben lecsupaszított, minimalizált, szinte gépi JavaScript kódot kapunk. Lássunk egy képet az eredményről (csak erős idegzetűeknek).



A GWT plugin azon túl, hogy a fordítást megteszi, tökéletes debug környezetet is biztosít számunka, azaz úgy debugolhatjuk kódunkat, mintha normál Java alkalmazást debugolnánk. A kód a böngészőben fut, és a GWT plugin részeként kapott webszerver létesít kapcsolatot a böngészőnkben futó GWT plugin, valamint az Ecipse-ben futó kódunk között. Töréspontokat rakhatunk, objektumokat vizsgálhatunk, minden teljesen elérhető, és működik. Bár nagy rajongója vagyok a Firebugnak, JavaScriptben ilyen szintű debug tulajdonságokkal még nem találkoztam. Mindezen rendszer mögött a Google neve van, mely talán valamekkora biztosíték hosszú távra. Ezért döntöttünk hát a GWT mellett.

Persze nem minden fenékig tejfel. Bár a JRE nagy része támogatott, és ez a támogatás folyamatosan növekszik[7], egy natív Java alkalmazás ennél sokkal többet használ. Ha olyan kódrészlettel találkozunk, mely nem támogatott, a lehetőségeink a következőek:

- „Google a barátunk”. Rákeresünk, hátha valaki már megírta, és átrakta GWT-re.

[6] `http://code.google.com/intl/hu-HU/webtoolkit/`
[7] `http://code.google.com/intl/hu-HU/webtoolkit/ doc/latest/RefJreEmulation.html`

- Google még mindig a barátunk. Ha nincs GWT-ben, megnézzük, van-e Javában, és átalakítjuk a kódot, hogy kompatibilis legyen GWT-ben is. (Itt olyan fellelhető, szabad Java kódimplementációkra gondolok, mint például az OpenJDK).

- Megírjuk mi magunk Javában.

- Megírjuk JavaScriptben. Ezt a JSNI (JavaScript Native Interface) GWT funkcionalitás biztosítja, ám nem hiába hagytam ezt utoljára: a tiszta JavaScript kód nem lesz optimalizálva, és nem tudjuk debugolni Eclipse-ben. Mindettől függetlenül sokat használtam én is olyan esetekben, ahol implementálatlan HTML5 funkciókat kellett belefejleszteni a projektbe (<canvas> tag, drag and drop, touch events, külső JavaScript könyvtár – pl. MathJax használata – , stb.). Érdemes mindig figyelni az új frissítéseket, mert a funkcionalitás folyamatosan bővül, így most már mind a <canvas> tag, mind a touch events natívan implementálva van a GWT-ben, és ilyenkor érdemes létező JavaScript kódunkat átrakni (mint ahogy ez meg is történt).

Az, hogy hogyan történjen a portolás, szintén érdekes kérdés. Választhatunk, hogy kódot portolunk vagy funkcionalitást. Ha kódot portolunk, akkor talán érdemes függőségi vizsgálatot csinálni a Java kódban, és azokkal az osztályokkal kezdeni, melyek a legkevésbé dependensek, így nem veszünk el a hibajelzések között (annyira). Ha funkcionalitást portolunk, mint ahogy én is tettem ennél a projektnél (először csak egy egyenest szerettem volna rajzolni két pontra), akkor elkezdhetjük a funkcionalitáshoz szükséges kódok portolását, így előrehaladni. Készüljünk fel, hogy nem fog gyorsan menni nagy projekteknél: nálam az Eclipse 6000 hibajelzést dobott fel, mikor az első Java package-eket átmásoltam az új GeogebraMobile GWT projektbe.

Fontos, hogy a GWT ízig-vérig Web projekt. Teljes mértékben úgy használhatjuk, mint szeretett függvénykönyvtárunkat (pl. jQuery), megírhatjuk a HTML-t, stílust CSS-sel, és csak a JavaScriptet írjuk – Javában. Tehát nem Java projektet írunk, hanem Web projektet.

A GeoGebra egy MVC rendszer. A modell a kernel package, mely tartalmazza a GeoElemek (pontoktól az alakzatokig, függvények, szöveg, képek, csúszka stb.) osztályait és algoritmusait. Az EuclidianControler.java a kontroller, mely összeköti a kernelt az EuclidianView.java-val (ez a view), mely gyakorlatilag egy Jpanel. A konkrét rajzolás a java.awt.Graphics2d osztállyal történik.

Igazából magam is meglepődtem, mennyi hasonlóság van a Java kód működése és a GWT-s lehetőségek között. Mindkettő eseményvezérelt kód, és a Graphics2d csomag metódusai szinte 100%-ig megegyeztek a GWT <canvas> tag implementációjával. Így, bár voltak nehézségek, a portolás technikailag lehetséges volt, és szépen haladt. Legfontosabb trükkök: HTML5 File API és Drag&Drop használata a .ggb fájlok megjelenítéséhez (csak behúzzuk az oldalba, teljesen kliens oldali unzip – a JSXGraph JavaScriptben megírt kitömörítő kódjával), MathJax használata a LaTeX egyenletek megjelenítéséhez.

Több hónap telt el, megjelentek az első iPadek. Markus izgatottan írt egy konferenciáról, ahol csak úgy mellékesen bemutatta a GeoGebraMobile-t: „Gabor, your work is very important. Everyone is speaking about iPad and Android here". Ekkor még két kollégám szállt be a fejlesztésbe, így már hárman vittük tovább a projektet.

Azonban mindez mégsem tökéletes. Mivel gyakorlatilag másoljuk az eredeti kódot, ezért így mindig két kódbázis létezett: a GeoGebra, és emögött elmaradva a GeoGebraMobile. Egyre többször és többször kaptunk megjegyzéseket, hogy „hú, ez nagyon tuti, de jó lenne, ha támogatná ezt a funkciót is, mint az eredeti…". Itt most csak a kernel funkcionalitásról beszélek, tehát egyes GeoElemek vagy kapcsolataik nem megfelelően vagy még egyáltalán nem működtek a mobil verzióban. Ahogy közeledett a 4.0-s GeoGebra verzió, a fejlesztők egyre többet commitoltak, mi pedig egyre jobban elmaradtunk. Nyáron a GSoC (Google Summer of Code) is támogatta a fejlesztést, így egy negyedik fejlesztő GUI kódot is rakott a GeoGebraMobile-ba (amely addig csak megjelenítésre volt

képes, lásd fent a képet), melynek eredményeképp 3 különböző kódbázis lett. Ekkor már láttuk, hogy ez így nem lesz jó.

Mindenesetre eljutottunk odáig, hogy ha a GeoGebraTube[8] oldalt mobile böngészőben nyitjuk meg, a GeoGebraMobile JavaScript kód által feldolgozott szerkesztést kapjuk. Persze, mint írtam, vannak szerkesztések, melyek nem működnek, vagy csak részben, de 70%-os sikerrel járunk, és kipróbálhatjuk Android vagy iPad – iPhone – iPod Touch eszközeinken a szerkesztéseket. Ha nagyon akarjuk, még Kindle e-book olvasón is megnézhetjük.

3. GeogebraWeb

Augusztus végén, a nemzetközi GeoGebra konferencián összeültünk mi fejlesztők, és egyéb workshopok mellett azt is megvitattuk, mi lenne a legjobb. Arra jutottunk, hogy nem választhatjuk szét a két projektet, mivel most már láttuk, mekkora teret nyert a HTML5, a mobile eszközök, és nem mellékesen (sajnos) mennyire bukik a Java. A végeredmény-tervezet így szólt: három projektet kell vinnünk tovább. Azt a részt, mely nem kompatibilis a GWT-vel (GUI, Swing stb.), a desktop verzióba tesszük. A Desktop projekt függeni fog egy másik projekttől, mely a Common nevet kapja, és már GWT kompatibilis kódot tartalmaz. Aki csak az eredeti GeoGebra fejlesztését viszi, a továbbiakban ezen két projekttel kell csak foglalkozzon: a kódjának azt a részét, amely nem kompatibilis a GWT-vel, a Desktopba rakja, amelyik pedig az (ez nagyrészt a logikai rész, valamint az algoritmusok csoportja), megy a Commonba. A Commont néha leellenőrzi a GWT fordítóval, hogy nem került-e bele véletlenül inkompatibilis kód.

A Commontól függ a harmadik projekt is: a Web. Ez a régi GeoGebraMobile, ami a logikát (modellt) a Commonból veszi, a megjelenítést és egyéb webspecifikus dolgokat pedig önmaga implementálja. Az elmélet tehát megszületett, a kivitelezés folyamatban van.

3.1. Git vagy Svn

Tervezés közben többször felmerült a Git[9]. Mivel a GeoGebra open source projekt, a GitHub meglehetősen elterjedt, és maga a Git igen csábító funkcionalitással rendelkezik, úgy döntöttünk, teszteljük egy picit: Kovács Zoltán kollégámmal ráálltunk. Zoli munkájának eredménye egy folyamatosan Svnből frissülő repo lett a GitHubon[10], én pedig beleszerettem a lehetőségekbe, melyeket a portolásnál nyújt: a kézreálló branching-merging funkciókba, illetve abba, mennyire jó a kód és fájlkövetés, hogy a két legfontosabbat említsem. Hozzáteszem, mindketten parancssor-mániások vagyunk, így hát itt az Eclipse-t tényleg „csak mint szerkesztőt” használtam, és a Gittel a parancssorból játszottam.

Szóval mi beleszerettünk, de csak ketten vagyunk a mintegy 30 fejlesztőből,a többiek már megszokták és megszerették az Svn – Trac párosítást. Ahogy utánanéztünk, rájöttünk, hogy a Trac nem igazán implementált még Gittel (nyilván nincs is szükség rá, hisz a history a gépünkön van), és nem valószínű, hogy el tudjuk érni azt a funkcionalitást, amelyet már a begyakorolt Svn – Trac páros nyújt. Kiemelném, hogy ez nem a Git hibája, egyszerűen az a koncepcióváltás, hogy a webes felületről áttérjünk egy offline felületre, valószínűleg megölné a projektet (csak mi ketten dolgoznánk tovább :-)). Fokozza még a problémát az Eclipse EGit pluginja, mely „csak” annyira bugos néha, mint a Subversive Svn plugin. Mivel azt már megszokták, és senki nem akar parancssorból dolgozni, itt is alulmaradunk. Ráadásul azoknak, akik portolni akarnak a history követése érdekében (tehát mikor átrakunk egy fájlt a Desktopból a Commonba), a Subversive sem jó, mert nem Svn move-ot használ, hanem egyszerűen törli, majd újra létrehozza a fájlt, így eltűnik a history. A portoláshoz a

[8] http://www.geogebratube.org/
[9] http://git-scm.com/
[10] https://github.com/geogebra

Subclipse plugint kell használnunk, ami már megfelelően megteszi a bejegyzést a historyba, és így visszakövethető a portolt fájl is.

Egyelőre tehát dobtuk a Gitre való átállást, és csak a fanatikusok fognak vele dolgozni, lásd alább.

3.2. Fejlesztés menete – meglett végül is

Így hát maradtunk az Svn-nél, egyelőre. Ha idáig eljutottál, kedves olvasó, és felmerül benned, hogy fejlesztenél (annak nagyon örülnénk ám! :-)), az alábbi módokon teheted meg:

1. Svn jó nekem: Amennyiben ebbe a csoportba tartozol, csak el kell olvasnod a `http://www.geogebra.org/trac/wiki/SetUp` oldalt, és már mehet is!

2. Svn jó nekem, és a GeoGebraWeb is érdekel: A fenti wiki oldal elolvasása után a Common projekt mellett a Webet is le kell szedned. Erről még készül a wiki. Mindemellett a Subversive plugin helyett a Subclipse-t kell használnod, hogy a portolt fájl historyja megmaradjon.

3. Git fanatikus vagyok: Igazi álmom lenne, hogy a Git repo és az Svn repo két irányú szinkronban működne, tehát egy automatizmus időnként szinkronizálná a két repót, így bármelyikben történik fejlesztés, az látszana a másikban is. Ez valószínűleg tényleg álom. Ám ami biztos működik, az a git-svn. Az Svn repositoryt a git-svn paranccsal leszedjük, majd mehet a fejlesztés. Célszerű minél lineárisabb historyt fenntartani, hisz az Svn úgysem fog tudni másfélét kezelni. Ez azt jelenti, hogy a branchokban végzett munkák után rebase-t kell használni, valamint emészthető committá squasholni a commitokat (jó-jó, de ez egy szakmai cikk, nem? :-)), hogy az Svn-nek fogyasztásra alkalmas legyen. A következő feladat ezek Ezután a git svn rebase futtatása, majd git svn dcommit-tal felküldeni a változtatásainkat. A munka többi része persze mehet Eclipse-ben, de ezt parancssorból vagyunk kénytelenek megcsinálni, ami egy Git usernek úgyis elengedhetetlen.

4. Konklúzió

Ilyen megoldott és még megoldandó problémák vannak most a GeoGebra életében. Mivel ez az írás áttekintő jellegű volt, egyrészt a helyszűke, másrészt „ha az egyik momentumot kifejtem, miért pont azt fejtsem ki” hezitálásom miatt, bármi szakmai – nem szakmai kérdésed, kérlek, küldd a gabor@geogebra.org-ra. Örömmel fogok válaszolni, ha tudok; ha nem tudok, keresek olyat, aki tud.

Általános következtetésként pedig egy mondat: az bebizonyosodott, hogy Java projektek webre való portolása lehetséges, és megéri nekikezdeni.

Az nxlog naplózó rendszer

Botyánszki Botond

Kivonat

Napjaink informatikai rendszerein több tucat alkalmazás ömleszti magából az események napóit, ráadásul sok esetben teljesen más formátumban, operációs rendszeren és protokollon keresztül. A hatékony loggyűjtes alapvető feltétele annak, hogy ezek az események később kereshetőek, elemezhetőek és értékelhetőek legyenek.

Az nxlog egy olyan naplózó eszköz, amely segítségével több platformon tudunk logot gyűjteni, továbbítani és tárolni. Az apache stílusú konfigurációs formátumával és egyszerűen használható saját nyelvével egy olyan nyílt forráskódú eszközt kapunk, amely megpróbálja megteremteni az átjárhatóságot a különféle formátumok és protokollok között.

Tartalomjegyzék

1. Miért naplózzunk? 24

2. Az nxlog története 24

3. A feladat 24

4. Az nxlog képességeiről 26
4.1. Multi-platform 26
4.2. Erősen moduláris 26
4.3. Biztonságos működés 26
4.4. Többszálú párhuzamos feldolgozás 26
4.5. Egyszerű konfigurációs szintaxis és nyelv 27

5. Modulok 27

6. Gyűjtsünk logot végre 29

7. Mezők 30

8. Összefoglaló 32

1. Miért naplózzunk?

Napjaink informatikai rendszerein több tucat alkalmazás ömleszti magából az események naplóit, ráadásul sok esetben teljesen más formátumban, operációs rendszeren és protokollon keresztül. A hatékony loggyűjtes alapvető feltétele annak, hogy ezek az események később kereshetőek, elemezhetőek és értékelhetőek legyenek. Ez a probléma mára már kulcsfontosságúvá vált, ezért a kérdés nem is arról szól, hogy kell-e naplózni, hanem hogy mivel és hogyan.

2. Az nxlog története

A cél egy multiplatformos napló gyűjtő és menedzsment eszköz létrehozása volt, amely minimális erőforrásigény mellett nagy teljesítményre képes és a syslog protokollon túl több naplóformátumot és forrást is képes kezelni. Ennek megvalósítására több lehetőség is kínálkozott, többek között az rsyslog, a syslog-ng és az msyslog továbbfejlesztése. A legnagyobb problémát ezeknek a Windows operációs rendszerre portolása, illetve az erős syslog protokollhoz való kötöttségük jelentette. A helyzet azóta némileg változott, de abban az időben architekturálisan sem volt kielégítő ezek egyike sem. Nem voltak képesek többszálú feldolgozásra, a modularitás is csak az msyslog esetében volt adott. Az utóbbival tettünk is egy kísérletet néhány kifejlesztett modul formájában, azonban hamar be kellett látni, hogy a jelen kor követelményeit csak egy új szoftver segítségével fogjuk tudni kielégíteni. Így született meg az nxlog. Azóta több sikeres bevezetésen vagyunk túl ahol heterogén környezetben különféle Windows és Linux rendszerek naplóit gyűjti központosítottan, napi több millió esemény feldolgozása mellett.

Később úgy döntöttünk, hogy ezt elérhetővé tesszük az open-source közösség számára is, így a forráskódja – mely jelenleg körülbelül 90 000 sor C kód – 2011. október 14-én került publikálásra. Ez elérhető a SourceForge-on keresztül GPL/LGPL licenc alatt[1]. Itt a forráskódon kívül bináris csomagokat is közzétettünk Debian, Ubuntu, Red Hat és Windows operációs rendszerekhez.

3. A feladat

Ha szétnézünk a naplózás tengerén, akkor elképesztően változatos módszerekkel, formátumokkal, protokollokkal találjuk szembe magunkat. Először kukkantsunk be a kedvenc Linux rendszerünk `/var/log` könyvtárába. Ez az a hely ahova a rendszernaplók kerülnek. Itt a következőkkel találkozhatunk:

`/var/log/messages`

```
Oct 27 23:09:01 mephisto CRON[31358]: pam_unix(cron:session): session closed
                                                                for user root
```

Ezt a fájlt a syslog daemon írja, azonban a formátuma mégsem azonos a syslog protokolléval, mert a sorok elejéről hiányzik a priority érték amely a súlyosságot és a facility-t tartalmazza. A standard installációkon a syslog daemon úgy van konfigurálva, hogy ezek alapján szétválogatja különböző fájlokba a logokat. Húsz éve ez valószínűleg elegendő volt így, mára már véleményem szerint ez sok esetben csak hátrány.

`/var/log/dmesg`

```
[ 0.072001] Booting processor 1 APIC 0x1 ip 0x6000
```

[1] http://nxlog.org/

A formátuma ennek sem hasonlít a syslog daemon által írt állományokéra, de legalább nem bináris.

`/var/log/dpkg.log`

```
2011-10-12 19:22:50 startup archives install
```

Ez sem syslog...

`/var/log/wtmp`

Bináris formátum, ne is próbáljuk szöveges fájlként nézegetni.

`/var/log/apache2/access.log, /var/log/apache2/error.log`

Habár ezeket az Apache írja, a formátumuk még véletlenül sem azonos, de még csak az eddigiekre sem hasonlítanak.

Még oldalakon keresztül sorolhatnám ezeket a példákat amelyek csak a `/var/log` alatt találunk. Szinte minden naplófájl más, még az idő, és dátum formátumok sem azonosak egyikben sem. Természetesen ez halandó ember számára ez nem okoz különösebb problémát, hogy értelmezze a tartalmukat, de ha valamilyen elemzőbe be szeretnénk tölteni ezeket, akkor már nem olyan egyszerű a felolvastatásuk.

A fájlokból megszerezhető naplókon kívül még létezik számos más forrás, például különféle hálózati protokollokon keresztül is lehetőségünk van naplókat fogadni és küldeni. Ezen kívül létezik sok olyan alkalmazás, amely adatbázisba naplóz.

Ha a GNU/Linuxtól kicsit eltávolodunk és más elterjedt operációs rendszereken nézünk szét, akkor azt látjuk, hogy nem csak a szabad szoftveres közösség megy a saját feje után, hanem például a Microsoft Windows rendszerek alatt is számtalan különféle naplófájlformátummal találkozhatunk. Ezen kívül ott van még a sortörésre használt karakter, a különféle karakterkódolások, a többsoros eseménynaplókat tartalmazó állományok problémája. A Windows EventLog rendszernaplókat ráadásul nem is lehet közvetlenül olvasni, mivel ez egy bináris formátumú cirkuláris logfájl. Itt az operációs rendszer kínál egy API-t a naplóesemények lekérdezésére. Ennek legalább annyi előnye van, hogy nem a naplógyűjtőnek kell felolvasni az eseménynaplót, hanem ez már strukturált formában elérhető. Mindezeket még sorolhatnám, de valószínűleg sokunknak volt már dolga a változatosabbnál változatosabb naplókkal.

Természetesen ha csak egy fajta forrásból érkező eseményekre vagyunk kíváncsiak, akkor relatív egyszerű dolgunk van az adott formátum beolvasásával. Azonban ha az eseményeket centralizáltan összegyűjtve szeretnénk vizsgálni – amely ahhoz szükséges, hogy átfogó képet tudjon nyújtani az infrastruktúra működésének egészéről is – akkor sajnos ezzel a sokszínűséggel mindenképp kénytelenek leszünk megbirkózni ahhoz, hogy minden esemény naplóját el tudjuk hozni, majd tárolni.

Egyelőre vegyük a két leggyakoribb és legelterjedtebb naplózási módszert, a UNIX-os syslog-ot illetve a Windows EventLog-ot. Ha ezeket gyűjteni tudjuk, akkor az operációs rendszer eseménynaplóinak a jelentős részét le tudjuk fedni. A syslog esetében minden naplóüzenet tartalmazza az esemény idejét, a gép nevét, a súlyosságot, az alrendszert, az alkalmazás vagy folyamat nevét és az általa naplózott üzenetet. Az EventLog esetében ennél kicsit jobb a helyzet, mert itt néhány mezővel többet ad át nekünk az API, például a felhasználó nevét és a domain-t is. Azonban sok esetben ennél még több információra van szükségünk amikor bizonyos szempontok alapján megpróbálunk eseményeket keresni vagy kimutatásokat és elemzéseket futtatni. Például a hálózati kapcsolatok naplózásakor az üzenetben általában eltárolódik a forrás és a cél IP címe. Ha egy olyan kimutatásra lenne szükségünk amely összegzi, hogy mely szervereinkre milyen címekről érkeztek kapcsolatok, akkor ehhez az eseménynapló üzenet részéből kell előbányászni a szükséges adatot, mégpedig az IP címeket. Úgy gondolom ebből a példából is jól látszik, hogy nem elég a naplókat egyszerűen csak

eltárolni, hanem a legtöbb esetben bizonyos feldolgozásra is szükség van ahhoz, hogy hatékonyan tudjunk keresni, szűrni, riportokat készíteni és elemzéseket végezni. Ha csak natúr belehányjuk valamilyen konténerbe (adatbázisba vagy fájlokba), attól ez még nem garantálja, hogy a későbbiekben ez így ebben nyers formában elegendő lesz. Hurrá, hogy le van időpecsételve, tömörítve és még titkosítva is.

4. Az nxlog képességeiről

A fentiekből jól látszik, hogy komoly fába vágja a fejszéjét aki egy általános célú naplózó fejlesztésébe kezd. Az nxlog formájában igyekeztünk egy olyan eszközt megalkotni, amely hatékonyan képes kielégíteni ezeket az elvárásokat. A különféle formátumok és források támogatásán kívül essen még szó, hogy milyen képességeket láttunk fontosnak implementálni.

4.1. Multi-platform

Ahhoz, hogy egy heterogén környezetben különféle operációs rendszerektől beérkező naplókat gyűjteni lehessen, elengedhetetlen, hogy ezeken a rendszereken képes legyen futni és ne csak távolról tudjon logokat fogadni valamilyen hálózati protokollon keresztül. Ezáltal nem kell többféle naplózó szoftverrel küzdenünk. Az nxlog jelenleg a GNU/Linux, Microsoft Windows és HP- UX operációs rendszereket támogatja. Tervbe van véve más operációs rendszerekre portolása is és ez a tapasztalatok alapján várhatóan nem fog megoldhatatlan akadályokba ütközni.

4.2. Erősen moduláris

A körülményektől függően a legtöbb esetben nincs minden funkcióra szükség, így az nxlog csak azokat a modulokat tölti be, amelyre a konfigurációs állománya hivatkozik. Ezáltal az erőforrásigénye is jelentősen csökken. A modularitásnak és a C nyelvű implementációnak köszönhetően a memóriaigénye igen kevés. Hamarosan bővebben is lesz szó a jelenleg elérhető nxlog modulokról és az ezeket alkalmazó architektúrájáról.

4.3. Biztonságos működés

A naplózáshoz nincs feltétlenül szükség arra, hogy root jogosultságokkal fusson a program amely így komoly veszélyeket rejthet magában. Az nxlog képes arra, hogy a root jogoktól megszabadulva egy adott felhasználó jogaival fusson és így korlátozható a működése. GNU/Linux alatt szükség esetén képes capability-ket használni, ha bizonyos funkciók ezt megkövetelik.

4.4. Többszálú párhuzamos feldolgozás

Ma már a mobiltelefonokban is többmagos processzorok dolgoznak. Az nxlog képes egyszerre több processzort illetve processzormagot használni azáltal, hogy több szálon dolgozza fel a logokat. Ezen párhuzamosított működés nélkül előfordulhatna, hogy a kimeneti oldali leterhelés miatt nem lenne képes a bemeneten az UDP csomagokat kellő gyorsan fogadni, ez pedig csomagvesztéssel járna. Az nxlog jól skálázódik a hálózati kapcsolatok számának növekedésével, egyszerre képes akár több ezer kapcsolatot kezelni. A többszálú működésnek köszönhetően kiemelkedő teljesítményre képes, másodpercenként százezres nagyságrendben tud eseménynaplókat feldolgozni.

4.5. Egyszerű konfigurációs szintaxis és nyelv

Az nxlog Apache-stílusú konfigurációs fájlokat használ, mint ezt hamarosan látni fogjuk néhány példán keresztül. Ez a formátum igen elterjedt a nyílt forráskódú szoftverek körében ezért várhatóan nem fog akadályokba ütközni a megértése. Ezen kívül saját nyelve van amellyel nagyon rugalmasan testre szabható a kívánt naplófeldolgozás. Ez egy egyszerű programnyelv amely leginkább a Perlhez hasonlít. Akinek volt már dolga naplófeldolgozással az jó eséllyel ismeri a Perlt is, ezért tartottuk jónak bevezetni ezt a szintaxist valamilyen nehezen érthető makró-nyelv vagy sablon rendszer helyett.

5. Modulok

Az nxlog négy fajta modult képes használni: input, processor, output, extension. Az első három közvetlenül érintkezik a naplóüzenetekkel és ezeket össze kell láncolni egy útvonalba annak érdekében, hogy bármiféle naplófeldolgozás történhessen. Szemléletesen a láncolás a következőként néz ki: input → [processor →] output.

Az extension modulok nem részei egyik feldolgozó láncnak sem, ezekkel az nxlog funkcionalitását lehet kibővíteni. Elsősorban az nxlog nyelvből meghívható speciális függvényekkel és eljárásokkal lehet kiterjeszteni a képességeit. Ilyenek lehetnek a parser-ek, konverziót megvalósító műveletek. Ennek további előnye az, hogy az I/O-t az input és output modulok transzparens módon képesek végezni, így például egy protokoll feldolgozáshoz nem szükséges megírni a hálózati kapcsolatok kezeléséért felelős kódot.

Az extension modulok az `xm_`, az input modulok az `im_`, a feldolgozó (processor) modulok a `pm_`, az output modulok pedig az `om_` prefixszel vannak ellátva.

Jelenleg a következő nxlog modulok léteznek:

`im_dbi` A libdbi adatbázis absztrakciós rutinkönyvtár segítségével ezzel a modullal adatbázisból tudunk naplót elhozni. A támogatott adatbázisok: MySQL, PostgreSQL, MSSQL, Sybase, Oracle, SQLite, Firebird.

`im_exec` Az im_exec modul a konfigurációs állományban megadott parancsot indítja el és ennek a kimenetét olvassa.

`im_file` Ezzel a modullal fájlból van lehetőségünk logot elhozni.

`im_internal` Az nxlog saját üzeneteit ezzel a modullal tudjuk az útvonalakba beküldeni.

`im_kernel` Ez a modul a Linux kernel üzeneteket olvassa.

`im_mark` A megadott időközönként periodikusan generál egy üzenetet ahogy ezt sokan megszokhatták syslogd esetében.

`im_mseventlog` A Microsoft Windows operációs rendszereken az EventLog rendszernaplót olvassa ez a modul.

`im_null` Ez a modul nem generál semmilyen logot, elsősorban tesztelési célra készült.

`im_ssl` TLS/SSL kapcsolatokat fogadva lehetőségünk van titkosított csatornán keresztül naplóüzeneteket fogadni.

`im_tcp` Titkosítatlan TCP kapcsolaton fogad naplóüzeneteket.

`im_udp` UDP protokollon keresztül képes naplóüzeneteket fogadni, mint például az RFC 3164-ben definiált syslog üzeneteket.

`im_uds` Unix Domain Socket-ről képes naplóüzeneteket olvasni, mint amilyen tipikusan a `/dev/log`.

`pm_blocker` A láncban előtte álló modult blokkolja. Elsősorban tesztelési célokra készült.

`pm_buffer` Memóriába illetve diszkre puffereli a naplóüzeneteket abban az esetben, ha a kimeneti oldal nem tudja elég gyorsan (vagy egyáltalán) küldeni vagy írni ezeket.

`pm_filter` Reguláris kifejezés segítségével tudunk naplóüzeneteket szűrni oly módon, hogy csak azokat engedi tovább, amelyekre illeszkedik a mintánk.

`pm_norepeat` Az egymás után ismétlődő azonos naplóüzeneteket összevonja egy "last message repeated x times" üzenetbe, így valamilyen szándékolt vagy hibás működés által előidézett DOS lehetősége csökkenthető.

`pm_null` Önmagában nem végez semmilyen feldolgozást, ám az nxlog nyelv segítségével végrehajtható különféle feldolgozás a rajta áthaladó üzeneteken, ezért nem teljesen haszontalan.

`pm_pattern` Egy XML alapú szabálybázist használ amelyben a mintaillesztési szabályokat definiálhatjuk. Mintacsoportok segítségével illetve adaptív működési algoritmusával jóval optimálisabb működésre képes, mintha ezt felsorolásszerű reguláris mintákkal próbálnánk megoldani.

`pm_transformer` Segítségével az nxlog nyelv használata nélkül tudunk CSV és syslog kimenetet olvasni, illetve a kimenetet ezekre a formátumokra alakítani.

`om_blocker` A segítségével beragadást tudunk szimulálni a kimeneti oldalon.

`om_dbi` A libdbi adatbázis absztrakciós rutinkönyvtár segítségével ezzel a modullal adatbázisba tudunk logolni.

`om_exec` A konfigurációs állományban megadott parancsot indítja el és ennek a bemenetére írja az üzeneteket.

`om_file` Segítségével fájlba lehet a naplóüzeneteket irányítani.

`om_null` A fogadott üzeneteket eldobja, mintha a `/dev/null`-ba írnánk.

`om_ssl` TLS/SSL kapcsolatot kezdeményezve lehetőségünk van titkosítva továbbítani a naplóüzeneteket.

`om_tcp` TCP kapcsolatot kezdeményezve továbbít naplóüzeneteket.

`om_udp` A BSD Syslog szabvány által is használt UDP protokollon tud naplóüzeneteket küldeni.

`om_uds` Segítségével Unix domain socket-re tudunk naplózni.

`xm_csv` Az nxlog nyelvből meghívható `parse_csv()` illetve `to_csv()` eljárásokat exportál. A CSV formátumát a modul paraméterezésével tudjuk megadni.

`xm_charconv` Kódkészlet konverzió végrehajtására alkalmas függvényt exportál és automatikus kódkészlet felismerésére is képes.

`xm_syslog` A BSD syslog protokollban használt sor-orientált üzenetek felolvasására és formázására alkalmas eljárásokat kínál.

`xm_exec` Két eljárást exportál, amelyekkel aszinkron és szinkron módon külső program végrehajtására van lehetőségünk.

6. Gyűjtsünk logot végre

Lássunk egy konkrét példát végre, hogy miként is tudunk centralizáltan naplót gyűjteni az nxlog segítségével. Tegyük fel, hogy a hálózatunkban van egy Windows-t futtató gép illetve egy Linux szerver amelyen a naplókat szeretnénk tárolni.

A Windows-os gép konfigurációs állománya a következőképpen néz ki:

```
define ROOT C:\Program Files (x86)\nxlog
define CERTDIR %ROOT%\cert
define CONFDIR %ROOT%\conf
define LOGDIR %ROOT%\data

Moduledir %ROOT%\modules
CacheDir %ROOT%\data
Pidfile %ROOT%\data\nxlog.pid
SpoolDir %ROOT%\data

<Input in>
    Module      im_mseventlog
</Input>

<Output out>
    Module      om_ssl
    Host        192.168.1.1
    CertFile    %CERTDIR%/client-cert.pem
    CertKeyFile %CERTDIR%/client-key.pem
    CAFile      %CERTDIR%/ca.pem
    Port        1514
</Output>

<Route 1>
    Path        in => out
</Route>
```

Ez egy viszonylag egyszerű konfiguráció, az nxlog az EventLog rendszernaplót olvassa és titkosított SSL csatornán továbbítja a linuxos gép felé. Az `om_ssl` modul működéséhez PEM formátumú tanúsítványokra van szükségünk. Ezek generálására itt nem térünk ki. Az openssl segítségével parancssorból is tudunk ilyeneket generálni, illetve több nyílt-forráskódú grafikus alkalmazás is létezik erre a célra, mint például a gnomint.

A Linux szerverünk beállítása pedig az alábbi:

```
define CERTDIR /var/lib/nxlog/cert

<Input sslin>
    Module      im_ssl
    CertFile    %CERTDIR%/server-cert.pem
    CertKeyFile %CERTDIR%/server-key.pem
    CAFile      %CERTDIR%/ca.pem
    Host        0.0.0.0
    Port        1514
</Input>

<Input udpin>
```

```
    Module  im_udp
    Port  514
</Input>

<Output winlog>
    Module  om_file
    File  '/var/log/win.log'
</Output>

<Output syslogfile>
    Module  om_file
    File  '/var/log/syslog.log'
</Output>

<Route sslwin>
    Path  sslin => winlog
</Route>

<Route syslog>
    Path  udpin => syslogfile
</Route>
```

A Windows felől SSL-en érkező naplóüzeneteket az `im_ssl` modul sslin példánya fogadja és ezeket a `win.log` állományba írja. Ezen kívül a szerver az 514-es UDP porton is fogad üzeneteket és ezeket közvetlenül a `syslog.log` fájlba írja. Mivel itt nem történik syslog protokoll szerinti értelmezés, ezért a kimeneti állományba is az eredeti üzenetek kerülnek, tehát syslogd által írt fájloktól eltérő módon a sorok elején a teljes syslog fejléc is látható. Természetesen ha szükséges, a `pm_transformer` modul segítségével a formátum a syslogd-nél megszokott formára hozható.

7. Mezők

Sok esetben szükség van arra, hogy a naplókat más formára hozzuk, rendszerezzük, szűrjük stb. Ez általában csak úgy oldható meg, ha az üzenet tartalmát vizsgáljuk. Ha az eredeti üzenet strukturált formában érkezik, mint pl. a Windows EventLog esetében, akkor már eleve több mező is tartozik az üzenethez. Ezt lehetőségünk van még tovább bontani vagy más mezőket hozzáadni az nxlog nyelvből illetve további modulok segítségével. A modulok dokumentációjában megtalálható, hogy melyik milyen mezőket tölt. A mezőkre bontást az alábbi példával szemléltetjük amelyben egy syslog formátumú naplóüzenet látható.

```
<30>Nov 21 11:40:27 log4ensics sshd[26459]: Accepted publickey for john from
                                                    192.168.1.1 port 41193 ssh2
```

Ezt a `parse_syslog_bsd()` eljárás vagy a `pm_transformer` modul segítségével felolvastatjuk, majd további mintaillesztéssel még néhány fontos mezőt kivágva a következő eredményt kapjuk:

```
AuthMethod            publickey
SourceIPAddress       192.168.1.1
AccountName           john
SyslogFacility        DAEMON
SyslogSeverity        INFO
Severity              INFO
EventTime             2009-11-21 11:40:27.0
Hostname              log4ensics
ProcessID             26459
SourceName            sshd
```

```
Message               Accepted publickey for john from 192.168.1.1 port 41193 ssh2
```

Ezek után lehetőségünk van rá, hogy a mezők alapján egyszerűen tudjunk szűrni vagy más logfeldolgozási döntést hozni. A mezők tipizáltak, a következő típusokat kezeli az nxlog: string, datetime, boolean, integer, ipv4address, ipv6address. Létezik egy speciális mező, a `$raw_event`. Ez a mező minden olyan esetben értéket kap, amikor a naplóüzenet nem strukturált formájú. A hálózati (ssl, tcp, udp) és a fájl modulok támogatják a bináris átviteli módot. Ezáltal elkerülhető, hogy a mezőket valamilyen (pl. CSV) konverzió segítségével kelljen megőriznünk. A mezőkre a dollár ($) jellel tudunk hivatkozni az nxlog nyelvből. Minden modul támogatja az Exec direktívát. Az ebben megadott kifejezést az nxlog a modulon áthaladó minden naplóüzenet esetén kiértékeli és végrehajtja.

Most lássunk egy példát, amelyben az nxlog nyelvi eszközeivel a logot mezőkre bontva, majd ezeket vizsgálva képesek vagyunk bonyolultabb feldolgozásra is. A feladat az Apache access.log naplójának szétválogatása úgy, hogy a kliens, illetve szerver oldali hibák naplói két külön fájlba kerüljenek. Ezt úgy tudjuk megtenni, hogy a HTTP 400 és 500-as kódtartományhoz tartozó üzeneteit szétválogatjuk az alábbi konfigurációs állományban látható módszerrel:

```
<Input access_log>
  Module im_file
  File   '/var/log/apache2/access.log'
  Exec   if $raw_event =~ /^(\S+) (\S+) (\S+) \[([^\]]+)\] \"(\S+) (.+)
                       HTTP.\d\.\d\" (\d+) (\S+) \"([^\"]+)\" \"([^\"]+)\"/\
         { \
           $Hostname = $1; \
           if $3 != '-' $AccountName = $3; \
           $EventTime = parsedate($4); \
           $HTTPMethod = $5; \
           $HTTPURL = $6; \
           $HTTPResponseStatus = integer($7); \
           $FileSize = $8; \
           $HTTPReferer = $9; \
           $HTTPUserAgent = $10; \
           }
</Input>

<Output client_error>
  Module om_file
  File   '/var/log/apache2/client_error.log'
  Exec   if ($HTTPResponseStatus < 400) or ($HTTPResponseStatus >= 500) drop();
</Output>

<Output server_error>
  Module om_file
  File   '/var/log/apache2/server_error.log'
  Exec   if ($HTTPResponseStatus < 500) drop();
</Output>

<Route apache>
  Path   access_log => client_error, server_error
</Route>
```

Az Apache `access.log` állományát soronként beolvastatjuk, majd egy reguláris kifejezés segítségével mezőkre bontjuk. A Perlhez hasonló szintaktikával ($+sorszám) tudunk a reguláris kifejezésben zárójelezéssel meghivatkozott substring értékét kiolvasni. Két kimenetünk van, ezekben eldobjuk azokat az üzeneteket, amelyek nem az odaillő HTTP válaszkódot tartalmazzák.

Az nxlog-processor nevű bináris egy hasznos eszköz lehet az olyan egyszeri feldolgozásnál, mint amilyen az előző példában bemutatott `access.log` szétválogatása, amennyiben a már meglévő logot kell feldolgozni. Ez főként fájl alapú input esetén használható jól. Az nxlog-processor a beme-

netről addig olvas, amíg az el nem fogy, majd ezután kilép. Így tehát az nxlog-tól eltérő módon nem daemon-ként fut amely állandóan figyeli a bemenetet.

Használata a fenti konfig esetén ilyen egyszerű:

```
nxlog-processor -c apachefilter.conf
```

8. Összefoglaló

Az nxlog egy olyan naplózó eszköz, amely segítségével több platformon tudunk logot gyűjteni, továbbítani és tárolni. Az apache stílusú konfigurációs formátumával és egyszerűen használható saját nyelvével egy olyan nyílt forráskódú eszközt kapunk, amely megpróbálja megteremteni az átjárhatóságot a különféle formátumok és protokollok között. Ezáltal lehetővé teszi, hogy open- source alapokon heterogén környezetben központosított naplózást valósítsunk meg vagy bármilyen más feldolgozást végezzünk a naplókon. Bemutattunk néhány példát a használatára, azonban sok fontos funkciójáról nem tudtunk szót ejteni, mint például a korrelációs elemzések, riasztások, naplófájl rotálás, mintaillesztéses adatbázis stb.

Amennyiben az előadás után megválaszolatlan kérdés merülne fel, vagy a dokumentációból nem világos valami, akkor kérlek írjatok az `info@nxlog.org` címre. Javaslatokat és ötleteket is szívesen fogadok.

A rendszergazda barát Linux

A Linux, amit egy Windows szakember is imádni fog

Czakó Krisztián

Kivonat

Ha Linuxról van szó, ritkán hallok semleges véleményt. Vannak a „Linux hívők”, akik magasztalják – és hajlamosak a rossz tulajdonságairól elfelejtkezni – és vannak a „Linux utálók”, akik csak az utóbbit emelik ki. A „hívők” érveit nem részletezném, pár főbb kiragadásával most megelégszem: megbízhatóság, függetlenség a szoftvergyártóktól, nulla licenc költség, nagyfokú rugalmasság és skálázhatóság. Az ellentábor véleménye persze nem ez. Két dolgot emelnek ki minden alkalommal, ha Linux kiszolgálóról van szó. Az egyik következik a másikból. Szerintük a Linux kiszolgáló bonyolult, nehezen kezelhető, így a bekerülési költsége (az a bizonyos TCO, tehát az egyszeri és üzemeltetési költségek együttvéve) sokkal magasabb, mint az alternatív, kereskedelmi rendszereké, melyek kezelése egyszerűbb.

Nem a fenti véleményekkel szeretnék most vitába szállni. Őszintén szólva többé- kevésbé egyetértek velük. Sokkal inkább egy olyan Linuxot szeretnék bemutatni, ami rendelkezik az ismert pozitív tulajdonságokkal és frappáns választ ad a vele szemben felhozott kifogásokra. Egy Linux szerver, amit pofon egyszerű telepíteni, beállítani és üzemeltetni. Annyira, hogy ha egy alapvető hálózati rendszergazdai ismeretekkel rendelkező (pl. Windows) szakembernek adjuk a kezébe, további tanulás és utánajárás nélkül azonnal teljes és kiválóan működő rendszert tud vele építeni.

Tartalomjegyzék

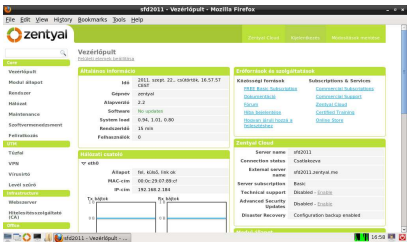
1. Hol kezdődött? 34
- 1.1. Miért tetszik nekem? . . . 34
- 1.2. Miért jó a Linuxnak? . . . 35
- 1.3. Miért jó a vevőnek? . . . 35

2. Mit tud a Zentyal? 35
- 2.1. A Zentyal telepítése . . . 36
- 2.2. Integrált rendszer . . . 36
- 2.3. Funkciók . . . 37
- 2.4. A motorháztető alatt, bővíthetőség . . . 41

3. Csatlakozz te is, és részesülj a Zentyal adta új lehetőségekből! 42

1. Hol kezdődött?

Amikor először találkoztam a Zentyal rendszerrel (leánykori nevén eBox Platform) megijesztett. Megijesztett, mert én is sokáig abból éltem, hogy értek a Linuxhoz míg mások nem. És a Linux TCO még így is jobb volt (na tessék, mégis vitatkozok a fentiekkel…), hát sokan megfizették a szakértelmet, cserébe jobb, összességében olcsóbb rendszert kaptak. A dolgok azonban megváltoztak, de erről majd később.



Szóval megijedtem, mert azt láttam ebben a rendszerben, hogy „elveszi a munkám”. Kicsit ipari forradalom érzetem lehetne, ha a bármi valós fogalmam lenne róla, hogy mit éreztek a munkások az ipari forradalom idején a gépekkel szemben. Aztán elgondolkoztam a dolgon, és arra kellett rájönnöm, hogy hülyeség a gépeket okolni. Inkább örülni kell az új lehetőségeknek (nyilván a munkások nem örültek a megnövekedett szabadidőnek, így a Linux rendszergazdától sem várható ez el, ha ez egyben a jövedelem arányos csökkenését is jelenti). De ha őszintén belegondolok abba, mit is vesz ez el tőlem, nem érzek nagy hiányt. A „hagyományos” Linux rendszerépítésben nem nagy kihívás ugyan azt az öt sort bemásolni az `smb.conf`-ba és a `slapd.conf`-ba. Ráadásul az esetek 99%-ban két szó kivételével pont ugyanazt. A hosztnév és az egyik „dc=” kivételével pont ugyanazt. Ha kicsit lustábbak vagyunk, akkor még ez sem különbözik.

1.1. Miért tetszik nekem?

A géprombolás tehát elvetve, már csak azért is, mert szoftvert nehezebb eredményesen rombolni mint fizikai gépeket. Nem maradt más, mint megbarátkozni az újjal és megnézni mi benne a jó. Az első ami egy Linuxosnak szimpatikus, hogy a Zentyal nyílt forrású. A második (ez már nem minden Linuxosnak lesz szimpatikus), hogy Ubuntu LTS-re épül. Azonban lássuk be, napjainkban új disztribúciót fejleszteni nagy merészség, de legalábbis komoly kihívás. Ezt a Zentyal fejlesztői is belátták és választottak. Sok lehetőségük nem volt. Ha nem kereskedelmi alapot keresünk, akkor a CentOS, Debian, Ubuntu hármasból lehet választani. Nem ismerve a döntés hátterét ebből a háromból az Ubuntu cseppet sem rossz választás. Én a CentOS-t a RHEL alapjai miatt vetném el (nincs önálló fejlesztés, a RHEL pedig egy önállóan is vállalati szegmenst célzó rendszer), a Debiant a bizonytalan kiadási ciklusok és a rövid távú támogatottság (új kiadás + 1 év az előző verzióra, de ki tudja ez mikor lesz?) miatt vetném el. Marad az Ubuntu, ami technológiailag sokat épít a Debianra, de garantált kiadási ciklusaival és a hosszú távú támogatással az egyes kiadások esetén (LTS szerver verzióknál 5 év) megfelelő alapnak tűnik egy vállalati szerver megoldáshoz.

A harmadik, és számomra legalább ennyire, ha nem még inkább fontos részlet az, hogy az egész megoldás szépen illeszkedik az Ubuntuba. Minden komponense önálló csomag (.deb) és minden egy önálló tárolóból (PPA) telepíthető. Nincs gányolás (make install vagy tar.gz kicsomagolás – sajnos sok ilyen megoldást láttam már), nincs egyedi csomagletöltögetés. Minden telepíthető apt-get segítségével is, ha épp úgy esik jól. Az én lelkivilágomnak mindenesetre jólesik, hogy ez a rész rendben van. Ugyanis ez az alapja annak, hogy egy rendszer stabilan üzemeltethető legyen.

Ha végül megnézem, hogyan fordíthatom a hasznomra azt, ami elsőre rémisztően hatott (az egész rendszer faék egyszerűségét) arra jutok, hogy végre valaki automatizálta és leegyszerűsítette azt a rutinmunkát, amit mi Linux szakemberek már – valljuk be – unalmasan, rutinból csinálunk.

1.2. Miért jó a Linuxnak?

Itt visszakanyarodnék az „ellentábor” fő érveihez: „a Linux bonyolult, a Linux szakemberek drágák és ezért a Linux megoldás TCO-ja valójában sokkal magasabb, mint a kereskedelmi megoldásoké”. A Zentyal esetén valami egész mással találjuk szembe magunkat. Egyszerű, átgondolt és integrált rendszert kapunk. A teljes beüzemelés és később az üzemeltetése egyszerűbb, mint egy Windows Small Business Server esetében. Megtartva a szabad szoftverek „árelőnyét” megszerzi az egyszerű bevezetés és üzemeltetés adta előnyt is.

1.3. Miért jó a vevőnek?

A Linuxot kedvelőket sosem kellett meggyőzni a rendszer előnyeiről. A piacon azonban a vevők igen jelentős többsége nem Linux kedvelő. Igazából még csak IT kedvelőnek sem mondanám őket, hiszen csak a bajuk van vele. Bonyolult, drága és nem értik a működését. Azt többnyire értik, hogy ez hogyan segít nekik (írógép helyett szövegszerkesztő, Magyar Posta helyett e-mail stb.), de elég sok bosszúságot is okoz. Kicsit elkanyarodva, pont ezt ismerte fel Steve Jobs és erre adott olyan választ, amivel az Apple ma ott tart ahol.

Visszatérve saját témánkhoz, a vevőt egyetlen dolog érdekli: a saját haszna. Ez így puritánul hangzik, de ettől még igaz. Egy cég azért vásárol meg bármit, mert azt elvárja tőle, hogy az a profitját növelni fogja. Teheti ezt a költségeinek lefaragásával (pl. olcsóbb rendszer vásárlásával) vagy a bevételeinek növelésével. Így vagy úgy az egyetlen dolog ami számít az a profit. Ha a Windows rendszer olcsóbb, működik (és már csak mi emlékszünk a ’90-es évekre, amikor nem működött), akkor azt fogja választani. Tehát vagy olyat adunk, ami olcsóbb, vagy olyat adunk ami sokkal többet nyújt ugyan azon az áron. A Zentyalban az a szép, hogy egyszerre képes mindkettőre.

A Zentyal azért ideális egy vállalatnak, mert alacsony bekerülési költsége mellett (ingyenes szoftver, olcsó telepítés, alacsony fenntartási költségek hosszú távon) magas szolgáltatás mennyiséget és minőséget kínál.

2. Mit tud a Zentyal?

Erre a kérdésre nehezebb válaszolni, mint gondolnád. Attól függ ugyanis, hogy ki kérdezi. Ha a vevőd kérdezi, akkor lehetőséget ad a költségcsökkentésre és a hatékonyság növelésére egy időben. Erről az előbb írtam. Ha egy IT szolgáltatás értékesítésével foglalkozó kérdezi, akkor új lehetőséget ad több olyan új ügyfél elérésére, akik eddig az alternatívák ára vagy tudása miatt nem vásároltak tőle. Vagy más megközelítésben, ha a meglévő ügyfeleinek ezentúl MS SBS helyett Zentyalt kínál, akkor többet tud adni olcsóbban úgy, hogy saját árrését közben növeli. Mutass egy vállalkozást, aki nem erről álmodik!

Ha informatikus kérdezi, akkor újabb két választ lehet adni. Linux szakembernek azt mondom, hogy lehetőséget a Linux eljuttatására oda, ahová eddig a bonyolultsága miatt nem sikerült eljuttatni. Lehetőséget arra, hogy kevesebb munkával több rendszert tudjon üzemeltetni. A szakmai tudása (a Linux belső rejtelmeinek megértése, ismerete) nem válik ezzel feleslegessé. Mindössze nem lesz rá nap mint nap szüksége.

Azoknak a szakembereknek, akik eddig nem dolgoztak Linuxszal, új lehetőséget nyújt. Számukra is megnyitja a nyílt rendszerek világát egyszerűségével anélkül, hogy hatalmas befektetésre lenne szükségük egy új rendszer – egy másik világnézet – megtanulására, megértésére.

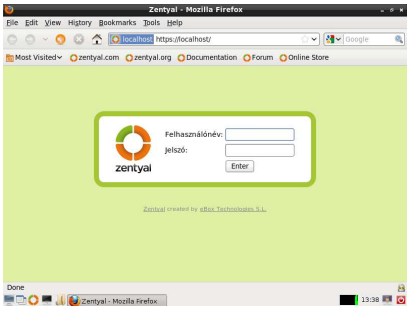
A továbbiakban az utóbbi két tábornak mutatom be a Zentyalt. Először a funkciókat, a beállítás egyszerűségét, a végén a technikai részleteket a Linux „szakiknak”.

2.1. A Zentyal telepítése

A telepítés legegyszerűbb módja, ha leöltöd a telepítő CD-t a `http://zentyal.org/downloads` címről. Itt találsz 32 és 64 bites változatot is. Én mai modern PC-ken a 64 bitest javaslom. Ha szereted a kihívásokat választhatod a telepítés alternatív módját is: az Ubuntu legfrissebb LTS kiadásából a szerver változatot kell telepítened. A telepítés után hozzá kell adnod a Zentyal PPA-t és telepítened kell a Zentyal alapcsomagot[1]

```
sudo apt-get install -y
python-software-properties
sudo add-apt-repository ppa:zentyal/2.2
sudo apt-get update
sudo apt-get install zentyal
```

Az eredmény mindkét esetben többé-kevésbé ugyanaz lesz. A Zentyal telepítőben van pár változtatás az eredeti Ubuntu LTS telepítőhöz képest, így lesznek eltérések.
Ezek egyike sem olyan, ami egy Linux-os szakembert zavarna, de ha nem vagy az, inkább maradj a Zentyal saját telepítőjénél. A „gyári” telepítő a Zentyal alapcsomagon kívül feltesz egy egyszerű grafikus felületet és a Firefox böngészőt. Az újraindítás után automatikusan bejelentkezik és elindítja a böngészőt, mely azonnal a Zentyal bejelentkező felületét nyitja. A további újraindítások során már az automatikus bejelentkezés elmarad, szükséged lesz a telepítéskor megadott felhasználónévre és jelszóra. Ha magad telepítetted Ubuntu LTS-ből indulva, a grafikus felület nem lesz ott automatikusan. Mindkét esetben igaz lesz viszont, hogy a szervered IP címén (vagy ha van DNS, akkor a nevén) https-sel eléred a kezelőfelületét, így a konzolra már nem lesz szükséged. A továbbiakban nem lesz eltérés a két telepítési módszer között.



Készítettem egy részletes videó tananyagot a Zentyal telepítéséről, melyet megnézhetsz a `http://zentyal.hu/` oldalon. A videóban lépésről lépésre megmutatom, hogyan érdemes a rendszert telepíteni.

2.2. Integrált rendszer

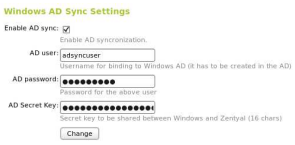
A Zentyalba bekerült minden olyan szolgáltatás, mely egy vállalati szervertől elvárható. Nyilvánvalóan ez a Linux adta lehetőségek egy töredékét jelenti csak. Ez természetes velejárója az egyszerűsítésnek. Nem korlát igazából, hiszen minden funkciót, amit egy „mezei” Ubuntu LTS szerveren meg tudunk valósítani, itt is elérünk ugyanúgy: parancssorból, kézzel (apt-get) telepítve.

Ami viszont bekerült, az teljesen integrálva lett a rendszerbe. Felejtsük el a kézi hegesztést, és a bonyolult beállításokat szolgáltatásonként. Az alap példánknál maradva, a Zenytalban a felhasz-

[1] Ha virtuális környezetben telepítetted a Zentyal kiszolgálót, a telepítést követően az első teendőd a virtualizációs környezet kiegészítőinek telepítése legyen VirtualBoxban:
`sudo apt-get install linux-headers-server virtualbox-ose-guest-utils`
VmWareben:
`sudo apt-get install open-vm-toolbox`

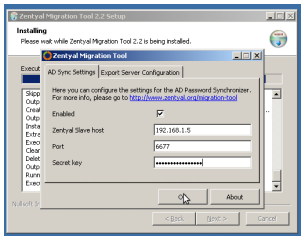
nálók kezelését LDAP címtárban oldották meg. Ez azt jelenti, hogy az OpenLDAP integrálva van a rendszerbe: ennek beállításaival minimálisan– akkor is csak haladó módban (ott beállíthatjuk a DN-t) – találkozunk. Egyszerűen megy. Az ehhez kapcsolódó funkciók (samba, dovecot, zarafa stb.) a telepítés után automatikusan megkapják az ehhez szükséges beállításokat. Ezzel nincs dolgunk és nem is találkozunk ezzel a felületen. Egyszóval nem kell minden egyes komponenst kézzel állítgatni, hogy működjön telepítés után. Feltesszük és megy.

A felhasználó kezelés területén három módban tudjuk használni a rendszert: az egyszerű és a mester mód ugyanazt jelenti: önálló kiszolgáló saját LDAP szerverrel és felhasználókkal, amihez később további Zentyal kiszolgálókat kapcsolhatunk szolga (slave) módban. A szolga mód kiválasztásával meg kell adnunk a mester kiszolgáló elérhetőségeit (ezt a mesteren az LDAP menüben megnézhetjük). Ezután a szinkronizáció elindul. A mesteren pedig nyomon követhetjük a szolga rendszerek állapotát.



A harmadik variáció egy érdekes, de mindenképpen hasznos megoldás. A Zentyal képes Microsoft Active Directory rendszerhez kapcsolódni és onnan venni a felhasználói adatokat anélkül, hogy ehhez Kerberost kellene telepíteni és beállítani.Ehhez nem kell mást tennünk, mint a Zentyal AD komponenst feltelepíteni az AD mester kiszolgálóra, felvenni ott egy szinkronizációs felhasználót és beállítani a komponenst. Az ott megadott adatokat és az AD címét megadjuk a Zentyal kiszolgálónknak és a felhasználók (jelszavaikkal együtt) szinkronizálva lesznek.

Ha a meglévő AD kiszolgálót szeretnénk kiváltani a Zentyallal, a Zentyal Migration Tool eszközkészlet áll rendelkezésünkre. Ennek két összetevője van: az egyik az a Windowson futó szoftver, mely képes exportálni a Windows Server beállításait (jelenleg Windows Server 2003 és 2008 verziókat támogat), valamint a Linux oldali megfelelője, mely az exportált adatokat importálni tudja. A Zentyal része egy teljes csoportunka szoftver is (a Zarafa, erről később lesz szó), mely szintén rendelkezik az Exchange szerveren (és Outlook PST fájlokban) tárolt levelek migrálásának lehetőségével. Ezzel együtt egy teljes eszköztár áll a rendelkezésünkre, mely segítségével egyszerűen és gyorsan válthatjuk ki régi, drága és megunt Windows szerverünket.



2.3. Funkciók

Fájlszerver, tartományvezérlő

Lássuk a Zentyal adta funkciókat. A rendszer alapját képező felhasználó kezelésről (LDAP, AD) volt szó az imént. Természetesen megtaláljuk benne a jól bevált Linux kiszolgáló szoftvereket. A fájlszerver és tartományvezérlő a samba. Annyit változtattak rajta, hogy alap telepítésben képes Windows 7 munkaállomásokat tartományba léptetni (a Windows 7 registry módosítása nélkül), beletették a valós idejű vírusellenőrzést (ClamAV-al) és a szemetes (RecycleBin) funkciót. Ezek természetesen megosztásonként ki- és bekapcsolhatók. A megosztások kezelését leegyszerűsítették. Mindent ott érünk el a kezelőfelületen, ahol logikus, azaz nincs külön „Fájlszerver” rész. Ha csoportnak akarunk megosztást készíteni, akkor a csoport jellemzőinél kell ezt megtennünk. Ha felhasználónak akkor ott. Ha olyan megosztást szeretnénk, aminél a jogokat külön be akarjuk tudni állítani, arra van egy külön menü. Adhatunk hozzáférést felhasználóknak és csoportoknak, ahogy a sambában ezt megszokhattuk.

Levelezés, csoportmunka

A levelezésért is a megszokott komponensek felelnek. A levelek fogadását és elküldését a Postfix végzi. Természetesen ennek a beállításai is a kezelőfelületről érhetők el, ahogy az integrált SPAM és vírusszűrésé is, melyekért az Amavis-ng, SpamAssassin és ClamAV hármas felel. A beérkezett levelek két helyre kerülhetnek. Ha „mezei" levelező rendszert készítünk, akkor a leveleink Maildir formában a fájlrendszerben kerülnek tárolásra, ahonnan a Dovecot szolgálja ki IMAP és POP3 protokollal. Természetesen bekapcsolható az SSL illetve STARTTLS is, amihez a rendszer részeként elérhető tanúsítvány- kiadót tudjuk segítségül hívni, ha nem akarunk kézzel hitelesíteni. A webmail sem hiányzik, erre a RoundCube áll a rendelkezésünkre, ami mögött egy Apache webszerver teljesít szolgálatot. Mondanom sem kell, hogy minden integráltan, és minden a kezelőfelületről állítható. Sem a parancssor, sem olyan extra tudást igénylő beállítások nem kerülnek elő, mely egy Linuxot még nem kellően ismerő rendszergazda számára nehézséget okozna. Ilyen probléma egyszerűen nincs.

Ha nem elégednénk meg a világ egyik legjobb levelező szerver környezetével, akkor leválthatjuk azt egy csoportmunka szoftverre. Erre a célra a nyílt forrású Zarafa kapott helyet a rendszerünkben. A Postfix/Amavis-ng/SpamAssassin/ClamAV csapat marad, csak a Dovecot/RoundCube részt cseréljük le. Pontosabban nem is kell lecserélni, mert virtuális tartományonként beállíthatjuk, hogy ki használja a Dovecotot és ki a Zarafát.

A Zarafa egy komplett csoportmunka megoldás. Tökéletesen alkalmas egy Exchange kiváltására. Rendelkezik minden szükséges funkcióval: levelezés, megosztható naptár, címjegyzék, teendők és jegyzetek. Nem hiányzik belőle az ActiveSync támogatás sem, így szinkronban tarthatjuk az adatainkat kedvenc okostelefonunkkal is. Amint korábban említettem, kapunk hozzá egy migrációs segédprogramot is, mely képes Exchange kiszolgálóról vagy tömeges Outlook PST fájlokból átemelni az adatokat. A Zarafa saját webes felülettel rendelkezik, mely rendkívüli mértékben emlékezteti a felhasználóz az Outlook Web Access (OWA) felületére. Ez sem lesz akadály a váltásnál. Természetesen kapunk IMAP és POP3 kiszolgálót is, így a megszokott levelezőprogramjainkat nem kell lecserélni. A naptárhoz pedig szabványos iCal és CalDAV felületen is hozzáférünk kívülről.

Ezek a nyílt forrású Zarafa részei. Ha Exchange kiváltására akarjuk használni meg kell emlékeznünk egy kereskedelmi komponensről is. Ez pedig az Outlook MAPI támogatás, melyhez két külön, zárt forrású szoftverre van szükségünk. Az egyik a Zarafa kiszolgálóhoz kell, melyet ingyenesen telepíthetünk (a Zentyal telepíti automatikusan) hozzá. Az ingyenes változat három párhuzamos Outlook klienst képes kiszolgálni MAPI-val. Ha ezen felül több munkaállomáson is szeretnénk Outlookot használni, meg kell vennünk legalább a Zarafa Small Business Edition licencét. Ezt (Zentyal esetében) a Zentyal forgalmazóknál tehetjük meg a legegyszerűbben. Aki ismeri az Exchange licencköltségeit most üljön le. A Zarafa Small Business változatának licencköltsége mindössze nettó 80€/5 felhasználó/év. Ez Egy Exchange licencnek a töredéke. Cserébe folyamatosan a legfrissebb Zarafa verziót használhatjuk és kapunk extraként egy külön mentő szoftvert hozzá, mely segítségével a visszaállításokat gyorsan és felhasználónként, azon belül mappánként végezhetjük el szükség esetén.

A Zarafa – főleg sok párhuzamosan dolgozó felhasználóval – jelentősen növeli a Zentyal hardverigényét, ami a kezelőfelület miatt eleve nagyobb, mint az ugyanarra képes csupasz Linux. Egy Zentyal szerverhez Zarafa nélkül 2 GB memória kell a gyors működéshez, Zarafával ezt minimum meg kell dupláznunk, de nagyobb felhasználószámnál a 8 GB is szükséges lehet. Ennek a fő oka, hogy a Zarafa MySQL adatbázisból dolgozik, melynek mint tudjuk (optimalizálva is) sokkal nagyobb a memóriaigénye mint az ext4 fájlrendszernek (Maildirben tárol levelek és adatok). Mindezekkel együtt is a Zentyalnak és a Zarafának a memóriaigénye messze elmarad egy Microsoft Windows Server és az Exchange igényeitől, azaz nem csak a licenceken, de a hardver költségeinken is faraghatunk. Arról az „apróságról" nem is beszélve, hogy ez nem OEM licenc, így ha szervert cseré-

lünk vihetjük magunkkal, nem kell újra megvenni (vagy nem kell eleve többszörös áron nem OEM licencet venni).

Hálózati infrastruktúra kezelése, hálózatvédelem, IDS, QoS, Proxy

Hétköznapi alapfunkciókról van szó, azaz az IP címek kiosztásáról (DHCP), a névfeloldás biztosításáról (DNS) és az idő szinkronizálásáról (NTP). Ezekre sorra az ISC DHCP és BIND kiszolgálóit valamint az alap NTP kiszolgálót kapjuk. Az érdekesség itt is az integrációban rejlik.

A kezelőfelületről vehetjük fel a domaineket a DNS kiszolgálóba és az IP tartományokat a DHCP-be. Ez eddig nem nagy kunszt. Az érdekesség ott kezdődik, hogy ha a DHCP esetén beállítjuk a DNS automatikus frissítését, akkor a DNS oldal beállítása automatikusan megtörténik (direkt és fordított feloldáshoz is). Ha mindemellett a web kiszolgáló funkciót is használni akarjuk, és annak beállításánál felveszünk egy új DNS domaint, a DNS kiszolgálónkba automatikusan bekerül. Nem kell két helyen beállítani ugyanazt.

Ugyanez az integráltság jellemző a tűzfal részre is. Alap telepítéssel is egy beállított tűzfalat kapunk, mely csak a szükséges funkciókhoz enged hozzáférést (a belső hálózatról is!). Ezen felül ha két csatolónk van, és az egyiket külsőnek jelöljük, akkor a külsőn letiltja a bejövő forgalmat, a belső felől pedig beállítja a címfordítást. Jobb alapbeállítást kapunk, mint a legtöbb SOHO router esetén. Ha emellett felteszünk egy új szolgáltatást, annak használatához automatikusan kinyitja a megfelelő portokat a belső hálózat felől és beteszi a szolgáltatások listájába a megfelelő port beállításokat, hogy ha módosítani akarunk ne nekünk kelljen keresgélni milyen portokat is kell kinyitni vagy bezárni. Nagyon hasznos.

Az alap Linux NetFilter tűzfalhoz kapunk Layer-7 támogatást is, előre elkészített mintákkal, így nem csak portokat adhatunk meg a szűrésnél, hanem protokollokat. Például tcp/80-as port helyett mondhatjuk, hogy a http forgalom. Ez sokkal általánosabb, mégis szigorúbb, hiszen így a 80-as porton sem mehet bármi, csak http. Alkalmazás szintű protokoll elemzést ne várjunk tőle, de az alap csomagszűrésnél sokkal fejlettebb megoldás.

A tűzfal beállításához, mint az az előzőekből igazából már kiderült objektumokat és szolgáltatásokat tudunk használni. Azaz a hálózati címeket (egyenként is) és tartományokat objektumokba rendezhetjük (akár többet is egybe) és ezen objektumokhoz rendelhetünk szabályokat. A szabályokban pedig nem portokat, hanem szolgáltatásokat adhatunk meg, ahol a szolgáltatásokba tudjuk felsorolni az oda tartozó portokat illetve protokollokat (Layer-7). A végeredménye egy átlátható, menedzselhető tűzfal lesz. A biztonságunkról ezen felül a Snort IDS gondoskodik, melyet csatolónként állíthatunk, ahogy azt is, mire figyeljen és mire ne.

Nem maradt ki a QoS sem. A szolgáltatásminőség garantálására a Linux kernel QoS rendszerét kapjuk természetesen. Ebben az esetben is ember számára emészthető formában, egyszerű beállíthatósággal. Csatolónkként, portra és protokollra szabályozhatjuk a sebességeket és a prioritást.

A rendszer képes kezelni a többirányú Internet kapcsolatot is. Pár kattintással beállíthatjuk a terhelés elosztását vagy hiba esetén az elsődleges irány váltását is valamint azt hogy miképp ellenőrizze a kapcsolatot (pl. átjárónak ping csomagok küldése). Aki próbált már ilyet Linux alatt az iproute-al beállítani az tudja értékelni, ha ezt pár kattintással el lehet végezni.

Kapunk egy proxy-cache szervert is (Squid), mely nem csak a böngészési élmény javítására alkalmas, hanem a Dansguardian segítségével a forgalom szűrésére is. Bekapcsolhatjuk a tartalom vírusszűrését (itt is a ClamAV fog szűrni) valamint beállíthatjuk a rendszer szűrési szigorúságát a tartalomra is. Ezt a nagyon megengedőtől a nagyon szigorúig öt fokozatban kapcsolhatjuk, ami mellett egyedi szabályokat vagy az Internetről letölthető szabálygyűjteményt tudunk használni. Lehetőségünk van a böngészési sávszélesség szabályozására is, mely a Squid erre a célra fejlesztett funkcióját használja. Érdemes vele „játszani”, mert nagymértékben tudja javítani a böngészési élményt terhelt vagy szűk sávszélességű vonalon.

Készíthetünk több szűrőprofilt is, melyeket aztán objektum vagy csoport- házirendhez kapcsolhatunk. Az objektum-házirendek a hálózati objektumok (azaz IP címek vagy tartományok) alapján állítja be a megfelelő profilt, míg a csoport-házirend az egyes felhasználói csoportokhoz rendeli azt. Ez utóbbi működéséhez be kell kapcsolnunk a felhasználó azonosítást is a proxynál, és ki kell kapcsolni a transzparens módot, hogy az azonosítás működjön. Egyéb esetekben a transzparens mód a kényelmesebb.

Távoli elérés, VPN

Az aktuális Zentyal verzió három VPN technológiát nyújt a rendszer és a hálózat elérésére illetve két Zentyal szerver (és a mögöttük lévő hálózatok) összekapcsolására. A méltán népszerű OpenVPN régi ismerős, már a korábbi kiadásokban is megtalálható volt. Ennek használatához szükségünk van egy saját hitelesítés szolgáltatóra (CA), mely a felhasználóinkat és a szervert is hitelesíti, hogy az férjen csak hozzá a rendszerhez, akit mi szeretnénk. Szerencsére a Zentyalban erre is van megoldás. A CA létrehozása pár kattintás, a szervezetünk neve, az országkód, a helység, a megye és a lejárati napok számának megadásával azonnal elkészül a cégünk saját belső használatra szánt tanúsítvány hitelesítője. Ennek birtokában elkezdhetjük felvenni és beállítani az OpenVPN hozzáféréseket. Először egy új VPN kiszolgálót kell létrehozni, majd azt beállítani. Ezt követően lehet tanúsítványokat készíteni a felhasználóinknak. A Zentyal nagyon ötletesen lehetővé teszi, hogy kész „VPN csomagot” töltsünk le, amit a felhasználónak odaadhatunk. Itt választhatunk Windows, Linux, Mac-OS és két Zentyal szerver összekötésére használható csomag között. Windows esetén az OpenVPN telepítőt is betehetjük a csomagba a beállítások és a tanúsítványok mellé.

Az IPSec támogatás a jelenlegi változatban még csak az osztott kulcsos hitelesítést támogatja, elsősorban két hálózat összekötésének céljából, amikor a túloldal szabványos IPSecet vár el. Ennek beállítása is egyszerű, az IP címek, a tartományok és az osztott kulcs megadása után működik a csatorna.

Windows felhasználók számára lehet kényelmes a PPTP támogatás, bár én ezt nem szoktam javasolni a megoldás ismert problémái miatt. Ha mégis szükség van rá, a beállítás itt is egyszerű. A VPN hálózat címtartományának (amiből a kliensek kapnak), a DNS és a WINS kiszolgáló címeinek megadásával a kiszolgáló kész. A bejelentkezéshez a biztonság érdekében itt külön felhasználókat kell felvegyünk, nem használhatjuk a központi azonosítást. Jobb is így.

Kommunikációs szolgáltatások

Itt két funkcióról érdemes szót ejteni. A szabványos XMPP üzenetküldő Jabber kiszolgáló is része a rendszernek, természetesen integrálva, így a bejelentkezéshez a rendszerben felvett felhasználói neveket és jelszavakat lehet használni. Külön érdekesség, hogy a belső felügyeleti rendszer is, ami monitorozza a szolgáltatásokat és erőforrásokat, képes XMPP-vel jelezni ha gondja van. Erre is használhatjuk a saját Jabber kiszolgálónkat (persze más XMPP szerverre is küldhetünk üzenetet).

Emellett egy alap telefonközpont funkciót is kapunk, mely az Asterisk PBX rendszerre épül. Itt csak a telefonáláshoz tényleg nélkülözhetetlen funkciók lettek a kezelőfelületen implementálva, (még) nem említhető egy lapon mondjuk a Trixboxszal. Ami benne van, az viszont korrektül illeszkedik a rendszerbe. Az alap beállításoknál felvehetünk egy külső szolgáltatót (szabványos SIP), amin keresztül a kimenő hívásokat intézhetjük és ahonnan a bejövő hívásokat fogadjuk. Emellett minden felhasználónkhoz rendelhetünk egy melléket (ezt a felhasználó jellemzőinél). Támogatja még a konferenciabeszélgetést is: felvehetünk „konferenciatermeket”, amibe a megfelelő teremszám és kód ismeretébe tudunk belépni („kívülről” is, ha van külső SIP kapcsolatunk). VoIP (SIP) telefonkészülékek használatához beállíthatunk nekik melléket, jelszót, a hangposta számát és egy értesítési e-mail címet, ahová levelet küldhet a rendszer, ha üzenetünk érkezett.

További hasznos holmik

Van még pár funkció, amikre itt most csak az említés szintjén térek ki. Ilyen az FTP kiszolgáló (vsftpd, integrálva, ahogy kell), a „captive portal” (Wifi hotspotok mögé), a Radius szerver (vállalati Wifi hitelesítéshez), nyomtatómegosztás (CUPS), „felhasználói sarok” (a felhasználóink beléphetnek és saját adataikat módosíthatják, ha van levelező szerverünk, akkor külső pop3 szerverről levél letöltést állíthatnak be) és virtualizáció (KVM alapú virtuális gépek létrehozása és menedzselése a Zentyal felületéről).

Fontosnak érzem itt megemlíteni, hogy a Zentyal fejlesztése folyamatos és nagyon gyors. A 2010. nyarán megjelent 2.0 és a 2011. ősszel megjelent 2.2 közötti időben hatalmasat fejlődött. A kezelőfelület gyorsabb lett, a funkciók nagymértékben bővültek. A teljesség igénye nélkül az újdonságok listáján volt a virtualizáció, az IPSec és a PPTP VPN, a Captive Portal is.

Mentés

A végére hagytam a minden rendszer kihagyhatatlan részét jelentő mentő funkciót. A Zentyal esetén két mentési részről beszélhetünk. Az egyik a beállítások és felhasználók mentése. Ez kevés adat, de ez kell a rendszer helyreállításához, ha azt újra kell telepíteni. Ezt exportálhatjuk egy fájlba magunknak, vagy az ingyenes felhő előfizetésünk segítségével tárolhatjuk központilag is. Utóbbiból a telepítéskor automatikusan helyreállíthatjuk a rendszerünket, vagy egy teljesen új kiszolgálót építhetünk egy alap beállításcsomagot felhasználva sokkal gyorsabban.

A mentés lényegi része természetesen az adatok mentése. Itt a jól ismert duplicity mentő szoftvert kapjuk. Ezt állíthatjuk be a kezelőfelületen. Használhatjuk helyi mentésre egy könyvtárba, vagy a helyi hálózaton rsync vagy FTP szerverre (e célra egy olcsó NAS kiváló megoldás lehet egy kisvállalatnál). De menthetünk a felhőbe is (Amazon, Zentyal cloud stb.). Utóbbi esetben a biztonságot a mentés titkosításával érhetjük el. A mentések időzíthetők, beállíthatjuk milyen gyakran kérünk teljes és milyen gyakran változás-mentést. A mentésekben kereshetünk és pár kattintással visszaállíthatjuk a hiányzó adatainkat.

2.4. A motorháztető alatt, bővíthetőség

Az elején említettem, hogy többek között azt szeretem a Zentyalban, hogy tisztán valósítja meg a funkcióit, illeszkedik az Ubuntu rendszerbe. Fontos azonban tudni, hogy ahhoz, hogy a rendszert kényelmesen, kellemes kezelőfelületről vezérelhessük kompromisszumokat kell kötnünk. Egy ilyen kompromisszum az, hogy a beállításokat (szöveges „konfig fájlok”) a rendszer felülírja. Ez – ha a megbízható működést tartjuk szem előtt – elengedhetetlen. Ha a normál (kezdő szintű) telepítést választottuk és az elején feltettük a komponenseket ez láthatatlanul megtörténik. Az alaptelepítéskor nem kiválasztott funkciók esetében azonban, az adott funkciót külön be kell kapcsolnunk, és ilyenkor egy listát kapunk azon beállítási fájlokról, amit felül fog írni. Itt aztán választhatunk, hogy akarjuk-e a Zentyal felületét az adott funkcióhoz használni vagy sem. Tiszta, egyértelmű megoldás.

Az egyes beállító fájlok elkészítéséhez a rendszer sablonokat használ a háttérben. Ezen sablonokat módosíthatod, így elvégezhetsz kézzel is olyan változtatásokat, amiket a webes kezelőfelület nem tesz lehetővé. Arra figyelj, hogy ne az eredeti sablont írd át (azt szó nélkül felül fogja írni egy esetleges frissítés). A Zentyalban – nagyon bölcsen – erre is felkészültek. A sablonok a `/usr/share/zentyal/stubs/modulneve/` mappákban találhatóak. A módosítandó sablont másold le a `/etc/zentyal/stubs/` könyvtárba (korábbi verzióknál a zenytal helyett még ebox volt a mappanév!). Amit oda lemásoltál az felülbírálja az eredetit, de a frissítéskor nem cseréli le a rendszer. Azzal számolj viszont, hogy ebben az esetben a sablon frissülése a gyári csomagban nem fog érvényesülni a változtatásod miatt. A stabil verzióban normálisan nem változnak a sablonok, de jobb odafigyelni erre.

A sablonok módosításán kívül lehetőséged van egyedi szkripteket futtatni az egyes modulok beállításakor. Ezeket a `/etc/zetyal/hooks/` könyvtárba kell tenned. A lehetőségről a Zentyal dokumentációjából és a Zentyal wikiben tudsz bővebben olvasni.

A Zentyal nem akarja átvenni az uralmat a géped felett mindenáron. Bármely általa támogatott funkciót kikapcsolhatsz, és akkor azzal nem fog foglalkozni, a beállításokat nem fogja generálni. Ugyanígy feltelepíthetsz az Ubuntu szerverre más komponenseket is, amik nem ütik a rendszer működését és beállíthatod, kezelheted azokat a „hagyományos" módon. Azaz a Zentyal kiválóan együttél a meglévő rendszerrel is, szakember számára lehetővé téve annak bővítését, megőrizve a Linux sokrétűségét. Eközben egy nagyon egyszerű, könnyen használható eszközt ad azok kezébe, akik nem értenek (adott esetben nem is akarnak érteni) a Linuxhoz, csak egy megbízható vállalati kiszolgálóra vágynak.

A Zentyal rendszerről magyarul találsz információkat a zentyal.hu oldalon. Fórummal, oktatóanyagokkal és tréningekkel igyekszünk segíteni a rendszer használatát. Amint azt a cikkben már említettem az oldalunkon találsz egy ingyenes telepítést oktató videót is. 2011. májusban az Ubuntu Developer Summit alkalmából Budapesten jártak a Zentyal fejlesztői is. Szerveztünk közösen egy szemináriumot, ahol Ignacio Correas (CEO) beszélt a rendszerről, Jorge Salamero Sanz bemutatta, hogyan használható a Zentyal Exchange kiváltására, én (Czakó Krisztián) pedig a Zentyal tűzfalként való használatát mutattam be. A szeminárium teljes felvételét is eléred a zentyal.hu oldalon.

3. Csatlakozz te is, és részesülj a Zentyal adta új lehetőségekből!

Azzal kezdtem a cikket, hogy bemutattam miért jó a Zentyal. Miért jó a Linuxnak, a végfelhasználónak. A Zentyal egy nagyon jó lehetőség azon informatikai vállalkozások számára is, akik egy jobb, olcsóbb alternatívát szeretnének kínálni ügyfeleiknek. Azoknak, akik több ügyfélre vágynak.

A Zentyal nyílt forrású és teljesen ingyenes. Ha csak ezt kihasználod, máris hatalmas lehetőségeket ragadsz meg. A Zentyalhoz kiegészítő szolgáltatásokat lehet vásárolni. Zentyal felhő előfizetést, ahol a szervereidet (vagy éppen az ügyfeleid szervereit) egy központi felületről menedzselheted, vagy éppen üzleti támogatást, hogy legyen biztosan szakember aki segít, ha valami nem úgy működik, ahogy szeretted volna.

Ha érdekelnek ezek a lehetőségek, és szeretnél te is profitálni ebből a kiemelkedő minőségű rendszerből, keress minket, regisztrálj a zentyal.hu partner oldalán! Mi előadásokkal, tréingekkel és szakmai anyagokkal segítünk neked abban, hogy többet, jobbat nyújthass ügyfeleidnek.

syslog-ng web GUI-k

Czanik Péter

Tartalomjegyzék

1. Mi a syslog? **44**
1.1. Miért syslog-ng? 44

2. Mire jó a központi log gyűjtés? **44**
2.1. Web GUI grep helyett?! 45

3. Syslog-ng web GUI-k **45**
3.1. Loggly 45
3.2. További felhő szolgáltatások 47
3.3. Loganalyzer 47
3.4. Logstash 48
3.5. Logzilla 49
3.6. SSB – Syslog-ng Store Box 50
3.7. ELSA: Enterprise Log Search and Archive 50

1. Mi a syslog?

A Wikipedia definíciója szerint a syslog egy szabvány a programüzenetek gyűjtésére. Lehetővé teszi, hogy el lehessen különíteni az üzenetet generáló programot az üzeneteket gyűjtő és feldolgozó alkalmazásoktól. Eredetileg a sendmail kiegészítőjének készült, de igen hamar rájöttek hogy ez sokkal általánosabban felhasználható. Így most már szinte minden operációs rendszer és hálózati eszköz támogatja a syslog használatát. Az eredeti megvalósítás fájlba és hálózatra tudta elküldeni az üzeneteket, az újabbak már adatbázisszervernek és még számtalan fogadó oldalnak képesek továbbítani őket.

1.1. Miért syslog-ng?

A syslog szabvány már nagyon sok éve velünk van, így számtalan alkalmazás született a megvalósítására. Ezek közül az egyik a syslog-ng. Hogy miért érdemes syslog-ng-t használni, ha egy átlagos Linux-disztribúcióban három különböző syslog megvalósítás is található? Csak néhány érv, fontossági sorrend nélkül:

- gyorsaság,
- nagy teljesítmény, függetlenül a konfiguráció hosszától, szűrőktől stb.
- sokféle forrás és cél: fájl, socket, pipe, program, TCP, UDP, titkosított (SSL), adatbázis (libdbi)
- patterndb: üzenetelemzés, akár korrelálás is
- makrók, sablonok, amikkel az üzenetek könnyen formázhatóak, külön válogathatóak stb.
- áttekinthető, logikus konfiguráció, ami pl. lehetővé teszi, hogy ha egy komplex szűrési feltételt több helyen is használunk, akkor azt elég legyen egy helyen definiálni és a többi helyen csak hivatkozni rá
- minden kiadáshoz elérhető részletes, jól érthető dokumentáció
- rengeteg platform támogatása: Linux, *BSD, HP-UX, AIX, Solaris stb.
- „made in Hungary”

Ezek közül önmagában bármelyik jelentős előny, de így kombinálva még inkább.

2. Mire jó a központi log gyűjtés?

A rendszerüzenetek központi gyűjtésének számtalan előnye van, kényelmi, praktikus, biztonsági és megfelelőségi is.

Kezdjük a kényelemmel. Ha nincsen központi log gyűjtés, akkor ha bármilyen problémát érzékelünk, akkor kénytelenek vagyunk bejelentkezni az adott gépre, megkeresni, hogy az adott eseményről vajon hova képződnek a logok, majd elkezdeni keresgélni benne. Ha az esemény több gépet is érint, akkor ugyanezt végig kell játszani további gépeken is. Ehhez be kell jelentkezni több gépre is, tudni kell a jelszavakat, a könyvtárstruktúrát stb. Központi log gyűjtés esetén egy helyen kell keresgélni az eseményeket egységes struktúrában, amivel időt és energiát takaríthatunk meg.

Az időt és energiát nem csak azzal takarítjuk meg, hogy kényelmesen hozzáférünk a logokhoz. Ha azt akarjuk megnézni, hogy mitől lassú az szerverünk, lehet, hogy már akkora a load, hogy percekig tart, amíg egyáltalán bejutunk a szerverre, majd a logok megjelenítése is nagyon lassan

megy. Vagy már be se jutunk, a konzolon sem. De központi loggolás esetén amíg az fsck-n dolgozik a gép, addig is már lehet keresni a hiba okát, mert a rendszernapló nem csak a vizsgált gépen található meg.

És persze biztonsági okokból is jó, ha a logok nem csak helyben vannak meg, ha nem egy másik helyen is. Ha megtörnek egy gépet, akkor az egyik első teendő a logok törlése és / vagy manipulálása, a nyomok eltüntetése részeként. Sokat viszont nem ér a helyi ügyeskedés, ha a központi log gyűjtő szerveren már ott van, hogy honnan és hogyan jutott be a támadó.

Ezzel eljutottunk a megfelelőséghez. A fentiek miatt nagyon sok szituációban a törvényi és egyéb szabályozások megkövetelik a központi log gyűjtést. Ilyenek például a PCI-DSS, COBIT vagy Magyarországon a HPT.

2.1. Web GUI grep helyett?!

Amiket idáig elmondtam a központi naplózásról, az jó esetben kevés Linux-felhasználónak lesz újdonság. Akinek meg az, remélem hogy ezek alapján belátja, hogy érdemes így csinálni. Viszont az előadás címében nem a központi log gyűjtés, hanem web GUI szerepel, ami már a hard core linuxosoknál kiverheti a biztosítékot. Miért kéne mindenféle színes, szagos, szélesvásznú webes felületeket használni, ott vannak a hagyományos linuxos eszközök, a grep és az awk, a gnuplottal pedig még grafikont is lehet rajzolni.

A grep, awk és társaik tényleg fantasztikus eszközök. De komplex kereséseknél vagy sok adatnál az SQL hátterű webes felületek sokkal gyorsabbak és hatékonyabbak. A bonyolult kereséseket pár egérkattintással össze lehet állítani, és ha tudjuk, hogy erre a későbbiekben is szükségünk lehet, akkor a legtöbb esetben a keresést el lehet menteni.

Ha másodpercenként több ezer üzenetünk érkezik be, könnyű eljutni a sok millió soros logokhoz. Az adatbázisok indexei ilyenkor is lehetővé teszik a Google- szerű válasz időket, azaz sok milliós adatsorból bonyolult lekérdezésekre is a másodperc tört része alatt választ kapunk. Ugyanez greppel akár két nagyságrenddel is lassabb lehet, 0,1s helyett 10s jellegű lekérdezési időket mértem kb. 5 000 000 üzenetnél.

Összességében a web GUI-k egyszerűsítik és gyorsítják a munkát, azaz sokkal gyorsabban lehet egy probléma végére járni, mint ha nem használnánk őket.

3. Syslog-ng web GUI-k

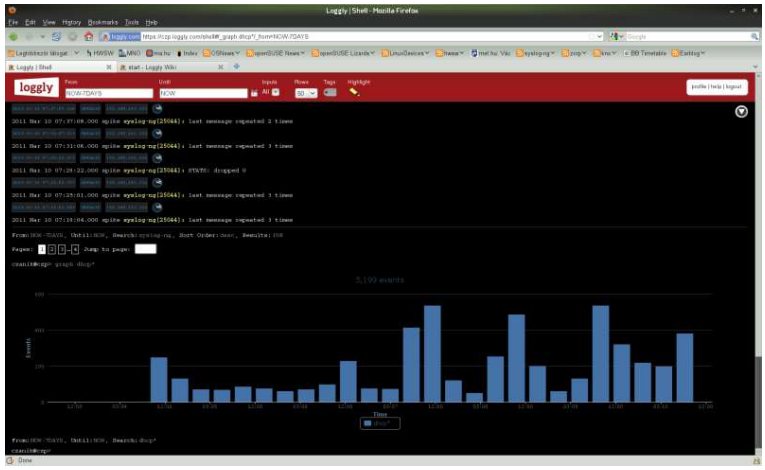
Most hogy már tudjuk, hogy miért érdemes a logjainkat központilag gyűjtenünk, és hogy webes felületen szeretnénk keresni bennük, ideje pár konkrét alkalmazást is megvizsgálni. Az elmúlt fél évben sok syslog-ng-n alapuló vagy syslog-ng-vel együtt működő alkalmazást kipróbáltam. Volt köztük a felhő szolgáltatásoktól kezdve az egyszerű php scripten át az extrém forgalomra felkészített rendszerekig minden. Ezekből emelem ki, amiket leginkább fontosnak éreztem, mutatom be röviden, sorolom fel előnyeit, hátrányait, ill. egy képernyőképpel is illusztrálom, hogy is néz ki.

3.1. Loggly

A Loggly egy felhő alapú szolgáltatás, azaz nincs szükség, hogy helyben bármilyen dedikált szervert telepítsen az ember. A használata nagyon egyszerű. Először is létre kell hozni egy felhasználói nevet a Loggly webes felületén. Bejelentkezés után egy varázsló megad minden információt ahhoz, hogy perceken belül már nézegethessük a beérkező logjainkat. Pár egérkattintás elég ahhoz, hogy a syslog-ng konfigurációba minimális módosítással bemásolható konfiguráció minta megjelenjen a képernyőn. Egyedül a log forrás átírására lehet szükségünk.

A Loggly üzleti alkalmazás, ahol az árazás függ a naponta beérkező üzenetek mennyiségétől és attól, hogy ezeket mennyi ideig szándékozzuk őrizgetni. A szolgáltatás bizonyos korlátok mellett ingyen is elérhető. Ezek szerencsére nem minőségi korlátok, azaz minden funkciója elérhető, de a napi maximális log mennyiség és a tárolási idő erősen limitált. Aki azonban csak egy-két kisebb gépet üzemeltet, valószínű jól el lesz ezekkel a korlátokkal is.

A webes felület nagyon kényelmes, letisztult. És hogy a hard core linuxosok is otthonosan érezzék magukat: parancssoros. Az első pár parancs összeállítása után már könnyen fog menni a lekérdezések és grafikonok készítése. A parancsok ezek után már könnyen előhívhatóak a historyból is.



Az elmúlt hetekben jelent meg egy új szolgáltatásuk is, az „Alert Birds”, ami lehetővé teszi, hogy ha bizonyos keresési feltételeknek megfelelő logok érkeznek be, akkor erről az Alert Birds figyelmeztetést küldjön. Ez lehet e-mail, flash alkalmazás, különböző API-k stb.

A Loggly ajánlható azoknak, akik új helyi rendszer üzembe helyezése nélkül gyorsan szeretnének központi log gyűjtő megoldáshoz jutni.

Előnyök:

- nincs szükség helyi szerver üzemeltetésére
- gyors és egyszerű használatba vétel
- UNIX életérzés :-)
- API a naplózáshoz, lekérdezéshez, konfiguráláshoz

Hátrányok:

- nem helyben van, azaz a hálózati kimaradások log vesztéshez vezethetnek
- az Internet kapcsolat gyakran túl drága nagy mennyiségű log feltöltéséhez
- a syslog üzenetek néhány része nem tárolódik, nem kereshető (pl. facility, severity)

További információ a Loggly honlapján[1] érhető el.

[1] `http://www.loggly.com/`

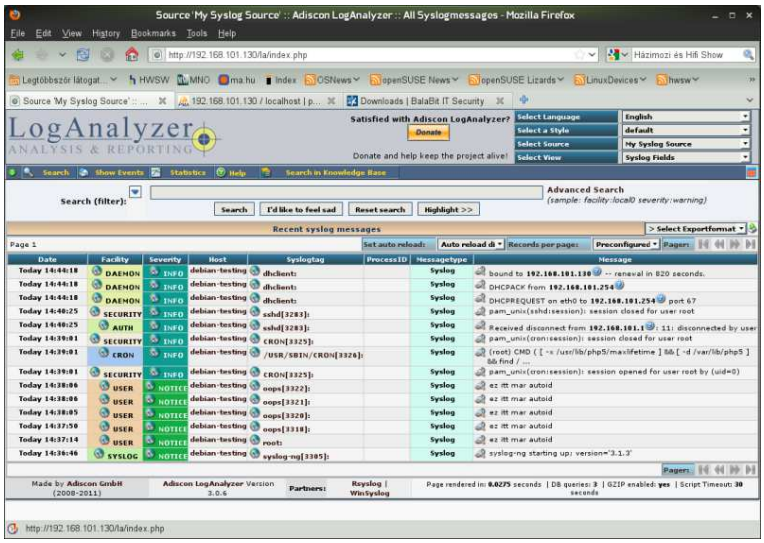
3.2. További felhő szolgáltatások

Az első kipróbálható felhős log szolgáltató a Loggly volt. Azóta továbbiak is megjelentek. Egyikük, a Logentries[2], amelyiket kipróbáltam, és sok érdekes technológia található benne, többek között saját log küldő függvények a legfontosabb programnyelvekhez, platformokhoz, keretrendszerekhez. A másik a PaperTrail[3], amelyiket még nem volt alkalmam kipróbálni.

3.3. Loganalyzer

A Loganalyzer egy egyszerű php alkalmazás, amely alkalmas a MySQL adatbázisba gyűjtött logok böngészésére és keresésére. Eredetileg nem a syslog-ng-hez készült, de pár apróbb korláttal (nincs külön mező a programnévnek és a processzazonosítónak), alkalmas syslog-ng-ből érkező logok megjelenítésére is. Minthogy MySQL-t használ a logok tárolására és indexelésére is, így nem túl jól skálázódik. A MySQL-be az adatok egyszerű „insert”-tel kerülnek be, ami másodpercenként pár száz üzenetnél többre nem elég és pár millió üzenetnél már a keresés is elég lassú.

A webes felület kicsit túlzsúfolt, rengeteg olyan információval, gombbal, adománygyűjtő linkkel stb. amikre a napi használatban nincs szükség, viszont a képernyő egy jelentős hányadát elfoglalják. De ha sikerült mindent jól beállítani és a képernyő közepére koncentrálni, akkor minimális syslog ismeretekkel és a dokumentáció bújása nélkül is könnyen használható.



A Loganalyzer egy egyszerű megoldás azoknak, akiknek nincs túl sok logja, és egy teljesen szabad webes felületen szeretne hozzáférni a logjaihoz.

Előny:

- egyszerű telepítés és webes konfiguráció

Hátrány:

- nem skálázódik

[2] `http://logentries.com/`
[3] `https://papertrailapp.com/`

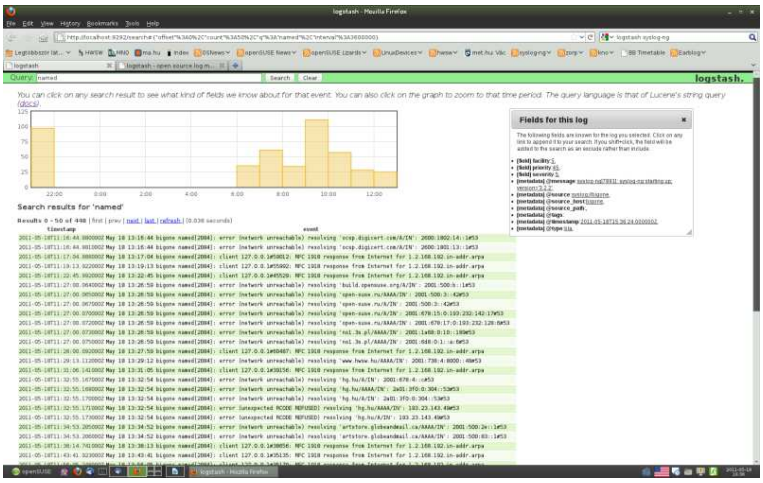
- zsúfolt képernyő, sok hiba

További információ a Loganalyzer honlapján[4] érhetőek el.

3.4. Logstash

A Logstash egy érdekes eszköz logok gyűjtésére, szűrésére és megjelenítésére. Sokféle forrásból, többek között a syslog-ng-től is tud logokat gyűjteni, különböző szűréseket, átalakításokat végezni rajtuk majd eltárolni egy adatbázisba és megjeleníteni webes felületen. A saját adatbázisán kívül még sok más kimenetre is el tudja juttatni a logokat.

Bár még mindig egy 1.0-ás verziójú szoftver annak minden korlátjával, de mégis ez az egyik legkönnyebben beüzemelhető webes felület. Elég egy fájlt letölteni, minta alapján elkészíteni egy pár soros konfigurációt és már gyűjti is a logokat fájlból vagy mint egy központi syslog kiszolgáló. A webes felület egyszerű, de nagyon könnyen és hatékonyan használható, a keresési történetek hiánya ellenére is.



MySQL helyett elasticsearch-öt használ, így sokkal jobban skálázódik, mint az egyszerű MySQL-es megoldások. Sok log esetén az adatbázis rész különválasztható akár külön gép(ek)re is, ami sokat segít a skálázódásban. Sajnos a log management még nem része a webes felületnek, így a régi logokat még csak az adatbázis-kiszolgáló közvetlen elérésével lehet kitakarítani. Ez azonban scriptekből könnyen automatizálható.

A Logstash ajánlható azoknak, akik percek alatt szeretnének üzembe helyezni egy jól skálázható rendszert és böngészni a logjaikat, majd később finomhangolni és bővíteni a rendszert.

Előnyök:

- egyszerű indulás, csak a sun-java a függősége
- jól skálázódik

[4] `http://loganalyzer.adiscon.com/`

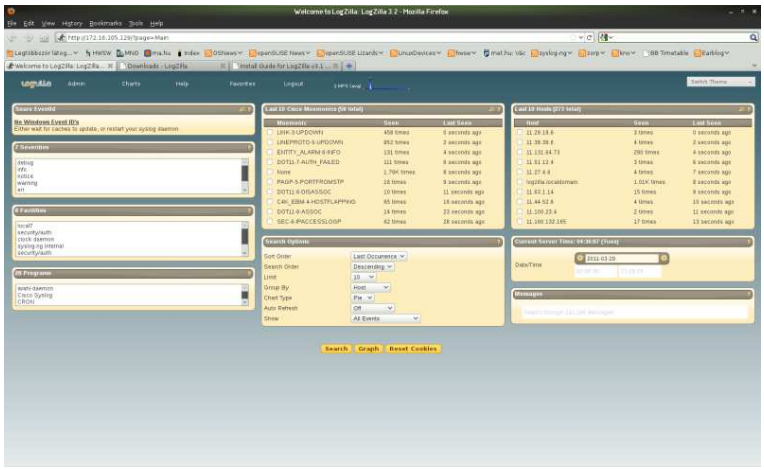
Hátrányok:

- nincs log management, hozzáférés szabályozás, mentett keresések

További információ a LogStash honlapján[5]

3.5. Logzilla

A Logzilla az egyik legrégebbi syslog-ng webfelület üzleti reinkarnációja, a php-syslog-ng-é. Felülete hasonlít az elődjére, de rengeteg új funkció belekerült. A felhasználói felület már támogatja a Cisco Mnemonics-ot, bővültek a grafikon készítési lehetőségek és már tud e-mail figyelmeztetéseket is küldeni. A háttérben már van LDAP integráció, üzenet deduplikálás, és külső indexelés hogy a nagy adatbázisok is gyorsan kereshetőek legyenek.



A Logzilla árazása rugalmas és sok szempontot figyelembe vesz. Így van pár kisebb korláttal rendelkező, de ingyen elérhető változata is, ami kisebb, Cisco eszközöket is használó hálózatoknak jó hír. Elérhető virtuális gépként is, ami bekapcsolás után szinte azonnal használható, de ahol nagyobb forgalom várható, ott ajánlott inkább fizikai gépre telepíteni. Ez persze sokkal több lépést igényel, de így sokkal jobb skálázódás érhető el.

A Logzilla ideális megoldás egy Cisco eszközökkel teli hálózati központba.

Előnyök:

- nagyon jó Cisco támogatás (Cisco alkalmazott fejleszti)
- nagy mennyiségű üzenet feldolgozása az optimalizált MySQL elérésnek és a sphinx-nek köszönhetően

Hátrányok:

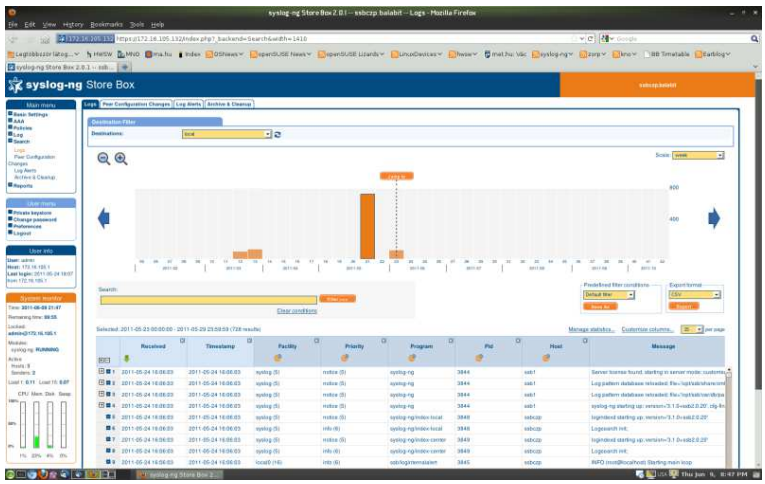
- fizikai gépre a telepítés nehézkes

További információ a Logzilla honlapján[6]

[5] `http://logstash.net/`
[6] `http://www.logzilla.pro/`

3.6. SSB – Syslog-ng Store Box

Az SSB egy naplózó szerver, amely alkalmas log üzenetek gyűjtésére, osztályozására, rendszerezésére és biztonságos tárolására. Egyaránt alkalmas üzemeltetési és megfelelőségi (compliance) célok megvalósítására. Része a syslog-ng kereskedelmi változata, mely több mint negyven platformról tudja továbbítani a logokat az SSB-nek. A webes felületen részletesen beállíthatóak a különböző felhasználhatói szerepkörök és a hozzáférés a logokhoz. Az SSB, mint a syslog-ng fejlesztői által készített webes felület, képes a legjobban kihasználni a syslog-ng kínálta lehetőségeket.



További információ az SSB honlapján[7].

3.7. ELSA: Enterprise Log Search and Archive

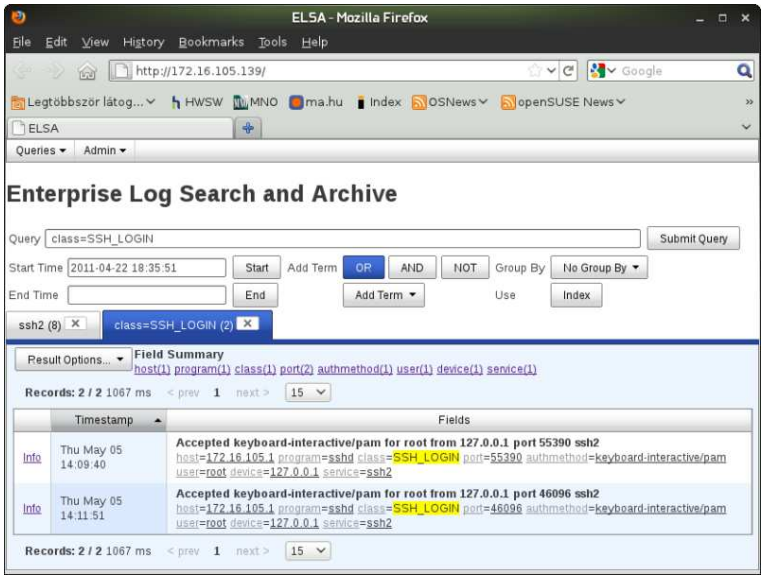
Az ELSA (ami egy angol rövidítés és szójáték: log archiválás és keresés vállalatoknak), egy új központi naplózó rendszer, mely a syslog-ng-n és a patterndb-n alapul. Az első komolyabb fejlesztés a BalaBiten kívül, amelyik kihasználja a patterndb-ben rejlő lehetőségeket. Az adattárolás és keresés a MySQL-re és a sphinx-re épül, melyhez egy egyszerű, de nagyon jól használható web felület csatlakozik, mely villámgyors elérést biztosít üzenetek millióihoz is.

Az ELSA új szoftver, még az 1.0-ás verziót sem érte el. Ennek velejárója, hogy szinte semmi dokumentáció nem elérhető, leginkább csak a Firefox támogatott, és azért van még mit finomítani rajta. Telepítése csak az Ubuntu 10.04-hez van leírva, de még oda is nehézkes. Jelenleg is folyik a következő verzió fejlesztése, ami szinte teljes újraírással jár, és remélhetőleg sokkal egyszerűbbé teszi a beüzemelést.

Architekturálisan az ELSA fel van készítve a folyamatos magas üzenetszámra, és hogy akár órákon át ennek többszörösét is fogadni tudja csúcs terhelésnél. A hasonló megoldásokban használt reguláris kifejezések helyett az ELSA patterndb- t használ, mely gyorsabb, és sokkal alacsonyabb az erőforrásigénye. Így képes a logokból a hasznos információt kinyerni, még hasznosabbá téve a beérkező üzeneteket. Például megkülönbözteti keresésnél, hogy egy adott IP-cím forrása vagy célja egy adott eseménynek, és nem csak úgy általánosan az IP-címre keres rá. Jelenleg a rendszer néhány Cisco, HTTP és Windows kapcsolódású patternt tartalmaz és további patternekkel bővíteni nem túl egyszerű. A következő verzióval várhatóan ez is könnyebb lesz.

[7] `http://www.balabit.com/network-security/syslog-ng/log-server-appliance`

A jobb skálázódás érdekében a webes felület, az adatbázis és az indexek külön gépekre is szétszórhatóak. Így érhető el, hogy extrém mennyiségű log üzenet is Google-szerű válaszidővel kereshető.



Az ELSA használata ott ajánlott, ahol extrém mennyiségű log képződik, és így minden kezdeti nehézséggel együtt is megéri az ezt kezelni képes szoftverrel foglalkozni. A tervek szerint a konferencia időpontjában már elérhető lesz egy új, sokkal könnyebben használatba vehető verzió.

Előnyök:

- Extrém skálázhatóság (15-20 kmsg/s folyamatos és 50-100 kmsg/s csúcs terhelés akár órákig)
- patterndb, amely jobban skálázódik a regexpes megoldásoknál és valós időben kigyűjti a hasznos információt

Hátrányok:

- korai fejlesztési fázisban, dokumentáció nélkül

További információ az ELSA honlapján[8] érhetőek el.

[8] `http://code.google.com/p/enterprise-log-search-and-archive/`

Haditudósítás a gigászok harcáról

Dr. Dudás Ágnes

Tartalomjegyzék

1. Történeti előzmények **54**

2. „Úgy kezdődött, hogy visszaütött” **54**
2.1. Hadüzenet a tengeren túl 54
2.2. Ellencsapás 55
2.3. További frontvonalak 55

3. Fegyverarzenál **57**
3.1. A Formaromboló 57
3.2. A Titkos Fegyverek 59

4. A FRAND-csel **60**

5. Diplomáciai hadviselés **61**

6. Hidegháború **61**

1. Történeti előzmények

A mobiltelefónia területén ismert módszer az iparjogvédelmi csatározás. Szinte minden jelentősebb konkurens hadilábon áll valakivel, az Apple többekkel is. Ezt már megszoktuk, de az, hogy a szabadalmi jogviták mellé kezd felzárkózni a design (formatervezési mintaoltalom, védjegy), az újdonság.

2. „Úgy kezdődött, hogy visszaütött”

2.1. Hadüzenet a tengeren túl

A történeti fonalat 2011 tavaszán érdemes felvenni. Április közepén indította az Apple az első támadásokat az Amerikai Egyesült Államokban a Samsung[1] és a Google ellen. A Samsung elleni – mellékleteivel együtt 373 oldalas – kereset[2] szabadalmi, formatervezési és versenyjogi elemeket is tartalmazott.

A kereset bevezetője szerint az Apple 2007-ben forradalmasította a telekommunikációs piacot az iPhone megjelenésével, amely többek között az érintőképernyő varázsát, a zenetárolási lehetőséget, az internetkapcsolatot illetve elegáns designt jelentette, és amely összetéveszthetetlen volt bármely más termékkel. 2007-ben felzárkózott mellé az iPod Touch és 2010-ben az iPad.

2007-ben a Time Magazin az év legjobb technikai kütyüjévé (Gadgets) választotta az iPhone-t. 2011 márciusára 108 millió iPhone került világszerte értékesítésre, iPodokból ez a szám meghaladja a 60 milliót, iPad esetében pedig a 19 milliót. További érdekesség, hogy az Apple reklámozási célra 2 milliárd dollárt költött 2007 és 2010 között. A benyújtott keresetben hivatkozik az Apple a megsértett szabadalmak listája[3] mellett formatervezési oltalmakat érintő bejelentésekre (design patents és trade dress)[4] valamint az egyes ikonokhoz tartozó védjegyekre[5] is. A jogsértő termékek között nevesíti a kereset a Samsung Captivate, Continuum, Vibrant, Galaxy S 4G, Epic 4G, Indulge, Mesmerize, Showcase, Fascinate, Nexus S, Gem, Transform, Intercept, az Acclaim smart phoneokat és a Samsung, Galaxy Tab tableteket is. Különlegessége a pernek, hogy az elj@?rás részletes dokumentációja online[6] is olvasható. Az ügy jelen állása szerint éppen szabadalmi szakértők meghallgatására kerül sor[7]

[1] Apple Inc. v. Samsung Electronics Co., 11-cv- 01846, U.S. District Court, Northern District of California (Oakland). http://www.bloomberg.com/news/2011-04-18/apple-files-suit-against-samsung-over-galaxy-phones-and-tablets.html

[2] http://www.apple.com/pr/pdf/110415samsungcomplaint.pdf

[3] 7,812,828 (the “828 patent”) Ellipse Fitting For Multi-Touch Surfaces, 7,669,134 (the “’134 patent”) Method and Apparatus For Displaying Information During An Instant Messaging Session, 6,493,002 (the “’002 patent”) Method and Apparatus for Displaying and Accessing Control and Status Information in a Computer System, 7,469,381 (the “’381 patent”) List Scrolling and Document Translation, Scaling and Rotation on a Touch-Screen Display, 7,844,915 (the “’915 patent”) Application Programming Interfaces for Scrolling Operations, 7,853,891 (the “’891 patent”) Method and Apparatus for Displaying a Window for a User Interface. 7,863,533 (the “’533 patent”) Cantilevered Push Button Having Multiple Contacts and Fulcrums

[4] D627,790 (the “’D790 patent”) Graphical User Interface For a Display Screen or Portion Thereof, D602,016 (the “’D016 patent”) Electronic Device, D618,677 (the “’D677 patent”) Electronic Device

[5] U.S. Registration No. 3,886,196, U.S. Registration No. 3,889,642, U.S. Registration No. 3,886,200, U.S. Registration No. 3,889,685, U.S. Registration No. 3,886,169, U.S. Registration No. 3,886,197

[6] http://dockets.justia.com/docket/california/candce/5:2011cv01846/239768/

[7] http://docs.justia.com/cases/federal/district-courts/california/candce/5:2011cv01846/239768/332/

2.2. Ellencsapás

A válaszlépésre nem kellett sokáig várni, a Samsung is kereseti kérelmet nyújtott be, amely elsődlegesen szabadalmi jogsértéseket nevesít[8].

A közzétett kereseti kérelemből szintén érdekes adatokat lehet megismerni: A Samsung 2005 és 2010 között 35 milliárd dollárt fordított kutatás-fejlesztésre és több, mint 50 000 kutató mérnököt foglalkoztatott 8700 telekommunikációs projektben. A Samsung szabadalmi portfóliója 2011. április 1-jén 28 700 amerikai szabadalomból állt, amelyből 5 933 a telekommunikáció területére tartozott.

1999-ben a Samsung jelent meg az okostelefonok első generációjával, amely egyesítette az internet-hozzáférés lehetőségét és a PDA funkciót. 2001-ben a Samsung PDA-ja (256 szín) nyerte a BusinessWeek „A legjobb termék" díját, illetve ezen évben állt elő először az ultraslim telefonnal, amely csak 9,8mm vastag volt.

A keresetlevél miután kitér még több mobiltechnológiai fejlesztéssel kapcsolatos adatra, részletesen bemutatja a jogsértés alapjául szolgáló szabadalmakat[9]

2.3. További frontvonalak

Ausztrália

Ausztráliában augusztusban nyújtott be keresetet az Apple, melyre reagálva a Samsung illetékesei azt nyilatkozták, hogy a Galaxy 10.1 Ausztráliában más formában kerül majd forgalomba, és így a formatervezési mintaoltalom megsértése fel sem merülhet[10]. Az ausztrál bírósági tárgyalásról egy blogon[11] tudósítottak „élőben", maga az eljárás meglehetősen nehézkes volt, tekintettel arra, hogy mindkét oldalon komoly érvek sorakoztak fel. Az ideiglenes intézkedést – amely keretében a forgalmazást betiltották – 2011. október 13-án hozta meg Bennett bírónő. Ezzel párhuzamosan 2011. szeptember 16-án a Samsung szintén keresetet nyújtott be az Apple ellen[12].

Japán

Japánban[13] a Samsung áprilisban megindított eljárására válaszul augusztus 23-án szintén a Galaxy szériák (S, SII) és a Tablet 7-es forgalmazásának betiltását kérte az Apple, arra hivatkozva, hogy mindezen termékek csak „szolgai másolatai" az iPhone, iPad termékeknek, ezen felül kártérítési igényét 100 millió jenben jelölte meg. Érdekessége az ügynek, hogy itt a bíróság szóvivője a folyamatban lévő eljárásról nem nyilatkozott. A 10.1-es Galaxy Tabok bevezetésére a per ellenére került sor. A forgalmazók nyilatkozata szerint a peres ügyet ezen a ponton nem tekintik a kereskedelmet akadályozónak.

[8] http://www.ibtimes.com/articles/139288/20110428/samsung-vs-apple-complaint-for-patent-infringement-full-text.htm

[9] U.S. Patent No. 7,675,941 ("the '941 patent"), U.S. Patent No. 7,362,867 ("the '867 patent"), U.S. Patent No. 7,447,516 ("the '516 patent"), U.S. Patent No. 7,200,792 ("the '792 patent"), U.S. Patent No. 7,386,001 ("the '001 patent"), U.S. Patent No. 7,050,410 ("the '410 patent"), U.S. Patent No. 6,928,604 ("the '604 patent"), U.S. Patent No. 6,292,179 ("the '179 patent"), U.S. Patent No. 7,009,626 ("the '626 patent"), and U.S. Patent No. 7,069,055 ("the '055 patent") (collectively "the patents-in-suit")

[10] http://ausdroid.net/2011/08/02/samsung-australias-official-comment-on-apples-complaint-to-the-federal-court/

[11] http://blogs.wsj.com/korearealtime/2011/09/29/live-blogging-apple-and-samsungs-hearing/?mod=yahoo_hs

[12] http://www.gizmodo.com.au/2011/10/apple-vs-samsung-the-injunction-stands/

[13] http://www.reuters.com/article/2011/09/08/us-apple-samsung-idUSTRE7871HH20110908

Dél-Korea

Dél-Koreában[14] is több szabadalom kapcsán állnak egymással perben a felek, jelenleg bizonyítási eljárások folynak. Az itteni eljárás következő érdemi fordulója 2011. november 25-én esedékes.

Európai Unió

Az Európai Unióban az első eljárások Németországban illetve Hollandiában indultak. Ennek oka egyszerű: a Németországban megindított eljárásból Hollandia azért marad ki, mert ott már korábban nyújtott be az Apple ideiglenes intézkedés iránti kérelmet (az ottani jogrendből következően a dél-koreai alperesre vonatkozó eltérő szabályok miatt), és egy tagállamon belül párhuzamosan futó ügyekre nincsen lehetőség.

Németország

Jelen esetben két alperes van, a német székhelyű leányvállalat ellen értelemszerűen lehet a német bíróság előtt pert indítani, vele szemben el is lehet járni, a másikkal (koreai anyavállalat) kapcsolatosan egy kiegészítő szabály alkalmazása szükséges: „Ha az alperes . . . székhelye nem a tagállamokban van, vagy a tagállamok egyikében sem rendelkezik telephellyel, az eljárást a felperes lakóhelye vagy székhelye szerinti tagállam bírósága előtt kell megindítani. . . ”[15] illetve kiegészítő szabályként azon tagállamok bíróságai előtt is, ahol a bitorlást elkövették, vagy megkísérelték.

A beadvány többek között a Német Szövetségi Bíróság „Meghosszabbított limuzinok”[16] esetére hivatkozik, amely értelmében bármely tagállamban megindított *közösségi*[17] formatervezési mintaoltalommal kapcsolatos – jogsértés abbahagyására vonatkozó igényt – az egész Európai Unióra vonatkozóan ki kell terjeszteni, tekintettel arra, hogy ha egy tagállamban jogsértő a cselekmény, fennáll a veszélye annak, hogy az EU más területén is az lesz.

A német eljárásban az Apple kereseti kérelme online hozzáférhető[18], a Samsung reakcióját pedig a médiában felbukkanó hírekből lehet rekonstruálni. Hollandián kívül a többi tagállam a keresetlevél szerint azért van „egy kalap alatt”, mert a Rendelet 82. cikke a joghatóságot az alábbiak szerint állapítja meg: „keresetek és igények tárgyában az eljárást az alperes lakóhelye vagy székhelye szerinti tagállam bírósága előtt kell megindítani, vagy – ha az alperes a tagállamokban lakóhellyel vagy székhellyel nem rendelkezik – az alperes telephelye szerinti tagállamban”. Alperesként pedig megjelölésre került a Samsung Gmbh és a Samsung Electronics Co. Ltd. Dél-Korea. A bíróság tárgyalás tartása nélkül – tekintettel a sürgősségi kérelemre, illetve az ideiglenes intézkedés szabályaira – elrendelte a Galaxy Tab 10.1-esek forgalmazásának betiltását az EU-n (kivéve Hollandia) belül 2011. augusztus 9-én. A bíróság 2011. augusztus 16-án átmeneti jelleggel a forgalmazás korlátozását Németország területére szűkítette.

A döntés megváltoztatásának okaként az derült ki, hogy az Apple beadványában – bizonyítékként szereplő – képek nem feleltek meg a valóságnak[19]

A német ügyben eddig csak ideiglenes intézkedések születtek (a holland bíróság időközben megállapította, hogy a benyújtott formatervezési mintaoltalom az oltalom feltételeit nem teljesíti, így

[14] http://fosspatents.blogspot.com/2011/09/apple-samsung-court-hearings-in-south.html

[15] 82. cikk (2) bek.

[16] BGH GRUR 2010, 718, 722. Rdnr 56,
http://juris.bundesgerichtshof.de/cgi-bin/rechtsprechung/document.py?Gericht=bgh&Art=en&az=I%20ZR%2089/08&nr=52206

[17] Ez a hangsúlyos elem, ha nemzeti mintaoltalmak lennének, minden tagállamban egyesével kellene lefolytatni az eljárásokat.

[18] http://www.scribd.com/doc/61993811/10-08-04-Apple-Motion-for-EU-Wide-Prel-Inj-Galaxy-Tab-10-1

[19] A tabletek összehasonlítása:
http://blog.webwereld.nl/wp-content/uploads/2011/08/Apples-Flawed-Evidence.jpg

az ezzel kapcsolatos kérelmeket elutasította, és a forgalmazás ideiglenes korlátozását csak szabadalomra való hivatkozással helyezte kilátásba), abból viszont több is, hiszen a Galaxy Tab 10.1 mellett a 7.7-es Tabok forgalmazását is be kellett szüntetnie a cégnek[20]

Hollandia

A holland bíróság 2011. augusztus 24-én helyt adott az Apple azon ideiglenes intézkedés iránti kérelmének, amely keretében a Samsung Galaxy S, Galaxy SII és Ace típusú okos telefonokat mindazon Európai országok vonatkozásában, ahol az EP 2059868 számon nyilvántartott (fotógalériával kapcsolatos) szabadalom érvényes[21]. Egyes szakértők[22] szerint viszont ezen szabadalom is tisztán szoftverszabadalomnak minősül, hiszen hardver oldalon nem tartalmazott feltalálói tevékenységet.

A holland döntés értelmében szeptember közepéig kellett a Samsungnak megoldást találnia arra, hogy miként válthatja ki azt a szoftverrészt, amely a szabadalmat sérti. A holland döntés egyébként csak fél sikert jelentett az Apple részére, nem állt meg ugyanis e bíróság előtt a keresetnek a formatervezéssel kapcsolatos ideiglenes intézkedés iránti igénye, ráadásul a bíróság – ahogy az később részletesen kifejtésre kerül – a „slide to unlock” szabadalom érvénytelenségét is megállapította.

A Samsung szintén pert indított a 3G technológiával kapcsolatos szabadalma megsértése miatt, amelyet azonban a holland bíróság – a későbbiekben hivatkozott FRAND elv alapján első fokon elutasított[23].

3. Fegyverarzenál

3.1. A Formaromboló

Az Apple által előszeretettel hivatkozott egyik csodafegyver egy formatervezési minta, amely a jelenleg is hatályos 6/2002/EK rendelet[24] (a továbbiakban: Rendelet) alapján: „a termék egészének vagy részének megjelenése, amelyet magának a terméknek, illetve a díszítésének külső jellegzetességei – különösen a rajzolat, a körvonalak, a színek, az alak, a felület, illetve az anyagok jellegzetességei – eredményeznek”. A mintaoltalom feltétele, hogy az új és egyéni jellegű (tájékozott használóra eltérő

[20] http://www.engadget.com/2011/09/04/apple-wins-german-injunction-against-samsung-galaxy-tab-7-7-pul/
[21] http://worldwide.espacenet.com/publicationDetails/description?CC=EP&NR=2059868A2&KC=A2&FT=D&date=20090520&DB=&locale=en_EP
[22] http://fosspatents.blogspot.com/2011/08/dutch-court-orders-eu-wide-preliminary.html
[23] http://fosspatents.blogspot.com/2011/10/samsung-loses-dutch-case-against-apple.html
[24] http://eur-lex.europa.eu/LexUriServ/site/hu/consleg/2002/R/02002R0006-20040501-hu.pdf

összbenyomást tevő) legyen[25]. Az Apple 2004-ben tett bejelentését 2010 februárjában módosította, az oltalom azonosítószáma: No 000181607-0001.
A hivatkozott rendelkezés tehát a rendelet 19. cikke, ami ezt mondja ki:

„A lajstromozott közösségi formatervezésiminta-oltalom alapján a mintaoltalom jogosultjának kizárólagos joga van a minta hasznosítására és arra, hogy a mintát engedélye nélkül hasznosító harmadik személyekkel szemben fellépjen. Hasznosításnak minősül különösen annak a terméknek az előállítása, forgalomba hozatalra való felkínálása, forgalomba hozatala, behozatala, kivitele, használata és e célokból való raktáron tartása, amelyre a mintát alkalmazzák, illetve amelyben a minta megtestesül."

Ezek tények:

1. Az Apple lajstromoztatta a formatervezési mintát.
2. Megjelent egy új piaci szereplő egy hasonló termékkel.

A per tehát arról szól, hogy elsődlegesen az Apple megpróbálja bebizonyítani, hogy a Samsung „bitorló".

Természetesen sokféle módon lehet védekezni, a legcélszerűbb – és szerintem a Samsung által is választott út (bár ez a beadvány sajnos nem áll rendelkezésemre) – egy, a formatervezési mintaoltalom megsemmisítésére irányuló viszontkereset lenne.

Megsemmisítési ok elég sok lehet, most csak a legjelentősebbre térnék ki, ez pedig a műszaki rendeltetés által meghatározott formatervezési minta. Nem részesülhet ugyanis oltalomban az a külső jellegzetesség, amely kizárólag a termék műszaki rendeltetésének következménye[26]. A formatervezési mintaoltalom természetesen grafikai illusztrációt tartalmaz, amelyet a beadványok több helyen idéznek is. Illetve a mintaoltalomból az Apple képviselői által kiemelt jellegzetességek[27] többek között az alábbiak:

1. egy négyszögletes termék, négy lekerekített sarokkal
2. egy sima, tiszta felszíni felület, amely a termék előoldalán található
3. egy látható fémkeret, amely a sima tiszta felszíni felületet határolja
4. egy képernyő, amely a tiszta felszíni felület alatt központosítva található
5. a tiszta felszíni felület alatt a képernyő egyértelmű, semleges lehatárolása minden oldalról
6. ha a termék be van kapcsolva, színes ikonok jelennek meg a képernyőn.

A fenti jellegzetességek mostanában már meglehetősen sok technikai eszközt körülírnak a mobiltelefonoktól az e-book readereken át a tabletekig.

Az Apple beadványa részletesen szemlélteti képek formájában a saját termékei, illetve az alperesi termékek hasonlóságait. Érdekes módon az egyik központi képen[28] a Galaxy 10.1 sokkal jobban hasonlít az iPadra, mint a Galaxy 10.1-re. A PCWorld-on olvasható erről egy részletes elemzés[29] is. (Ehhez érdemes még megjegyezni, hogy a beadvány szerint sem a forma „kisebb változtatása", sem a név feltüntetése a tableten nem minősülhet „érdeminek", tekintettel arra, hogy az összbenyomás azonos.)

[25] Rendelet 4. cikk (1) bek, és 6. cikk (1) bek.
[26] Rendelet 8. cikk
[27] Ideiglenes intézkedés iránti kérelem (`http://www.scribd.com/doc/61993811/10-08-04-Apple-Motion-for-EU-Wide-Prel-Inj-Galaxy-Tab-10-1`) 15-16. oldal, 1.2. pont
[28] Ideiglenes intézkedés iránti kérelem 24. oldal
[29] `http://www.pcworld.com/article/238047/apple_offers_flawed_evidence_in_lawsuit_against_samsung.html`

3.2. A Titkos Fegyverek

A titkos fegyvertár részét természetesen a szabadalmak képezik. Az eddig említett eljárások során is már több megemlítésre került közülük, terjedelmi korlátok miatt azonban csak néhány lényegesre térünk ki. Az egyik az érintőképernyő több pontos érintésével kapcsolatos[30], a másik pedig az operációs rendszer heurisztikus működési modelljét[31] védi. A probléma hasonló a korábban ismert szoftver alapú szabadalmak problémáihoz. Az Amerikai Egyesült Államok lehetőséget ad az ilyen jellegű bejelentésekre, viszont az is előfordulhat, hogy a találmány vizsgálatakor – az oltalom megadását megelőzően – a technológia állását nem mérik fel elég alaposan.

A holland bíróság például megállapította a „slide to unlock”[32] szabadalomról, amely az Európai Unióban[33] szintén oltalmat kapott, hogy az valójában a technika állásához képest[34] nem tartalmazott olyan mértékű újdonságot, amely az oltalomra alkalmassá tette volna.

Ekkor az Apple előállt a titkos fegyvernek számító „list scrolling and document translation, scaling, and rotation on a touch-screen display”[35] (a továbbiakban: 381-es szabadalom) című szabadalommal. A szabadalom érdekessége, hogy a leírás maga egyszerű (Szakértői szemmel nézve talán túl egyszerű is. Itt arra utalok, hogy a LaunchTile & AppLens „one-handed thumb use” technológiák[36] már 2005-ben is ismertek voltak.), viszont tökéletesen alkalmas lehet arra, hogy az okostelefon-piac konkurenseit lebénítsa. Ha ugyanis az eljáró bíróságok sorra megállapítják, hogy a szabadalom érvényes, az Apple viszont – szemben a korábbi gyakorlattal – nem ad díjfizetés fejében hasznosítási jogot, hanem a konkurens termékek forgalmazásának betiltását kéri, egyeduralkodóvá válhat.

A 381-es szabadalom érdekessége továbbá, hogy azt a Nokia már megkísérelte érvényteleníteni, azonban az általa megjelölt 20 pont egyike sem volt erre alkalmas. A Nokia vs Apple harci helyzetet legátláthatóbban talán Florian Müller foglalta össze[37].

Az Apple egyébként ezt a csodafegyvert használja a Samsung ellen Amerikában és Ausztráliában, valamint a HTC ellen indított eljárás egyik alapja is ez a szabadalom.

Amire senki nem számított az az, hogy a Motorola pert nyer a mannheimi tartományi bíróság előtt, mert az Apple elmulasztja az első tárgyalást. Ez azt jelenti, hogy a bíróság a felek meghallgatása és érdemi tárgyalás tartása nélkül a rendelkezésre álló dokumentumok alapján a keresetben foglaltaknak helyt ad. A magyar perjogban is ismert „bírósági meghagyás” azonban csak abban az esetben válik ítélet hatályúvá, ha az ellen az alperes 15 napon belül ellentmondással nem él.

---

[30] http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=7,663,607.PN.&OS=PN/7,663,607&RS=PN/7,663,607

[31] http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=7,479,949.PN.&OS=PN/7,479,949&RS=PN/7,479,949

[32] http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=7657849.PN.&OS=PN/7657849&RS=PN/7657849

[33] http://worldwide.espacenet.com/publicationDetails/biblio?DB=EPODOC&adjacent=true&locale=en_EP&FT=D&date=20080903&CC=EP&NR=1964022A1&KC=A1

[34] Neonode N1m bemutatása: http://www.youtube.com/watch?v=Tj-KS2kfIr0 2:53-tól a jobbról balra, balról jobbra mozdulatok ismertetése, a holland döntésről részletesebben: http://fosspatents.blogspot.com/2011/08/dutch-judge-considers-apples-slide-to.html

[35] http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=7,469,381.PN.&OS=PN/7,469,381&RS=PN/7,469,381

[36] http://video.google.com/videoplay?docid=-2417986546511555659

[37] http://fosspatents.blogspot.com/2011/06/apple-and-nokia-settle-patent-dispute.html, http://www.scribd.com/doc/52195210/NokiaVsApple-11-03-31-100

4. A FRAND-csel

Természetesen a Samsung maga is hatalmas szabadalmi portfólióval rendelkezik, így aztán elég könnyen talált a mérnökök és ügyvédek lelkes csapata néhány szabadalmat, amely bizony az Apple iPhone S4 ellen felhasználhatónak tűnt. Az egyik ilyen a 3G (UMTS) technológiai standard. Ezen a jogalapon pert indított a Samsung Amerikában és Olaszországban, Franciaországban és ahogy arra már utaltunk, Hollandiában is. Azonban a viszontkereseteiben az Apple a FRAND (Fair, Reasonable, Non-discriminatory) eljárásra hivatkozva előadta, hogy a szabványok alapjait képező szabadalmak esetében forgalmazástól való eltiltásnak nincsen helye, és így legfeljebb díjigénnyel állhatna elő a Samsung, viszont a termék által használt szabadalmakra vonatkozóan a kapcsolódó díjakat az Apple beszállítói még megfizették[38].

Azt, hogy az egyes eljáró bíróságok miként ítélik meg a helyzetet, nem tudni. Mégis nagyon érdekes maga a helyzet, amelyben jelenleg úgy tűnik, hogy a kisebb (kényelmi funkcionalitásokra szorítkozó) szabadalmi portfólió többet jelent, mint az, amelyik a technológiai fejlődés alapját adja, utóbbit ugyanis – a technológiai fejlődés érdekében vállalt egyezmény miatt – „meg kell osztani".

Nyilvánvalóan felmerülhetne itt is a szabadalmi kényszerengedélyek kérdése. A magyar szabadalmi törvény 32. §-a ugyanis kimondja, hogy:

> *„Ha a szabadalmazott találmány másik szabadalom (a továbbiakban: gátló szabadalom) megsértése nélkül nem hasznosítható, a függő szabadalom jogosultjának – kérelmére – a gátló szabadalom hasznosítására a szükséges terjedelemben kényszerengedélyt kell adni, feltéve, hogy a gátló szabadalom szerinti találmányhoz viszonyítva a függő szabadalom szerinti találmány számottevő gazdasági jelentőségű műszaki előrelépést jelent.*
>
> *(2) A gátló szabadalom jogosultja – ha a szabadalmára az (1) bekezdés alapján kényszerengedélyt adnak – a kényszerengedélyre vonatkozó közös szabályok szerint igényt tarthat arra, hogy méltányos feltételekkel engedélyt adjanak számára a függő szabadalom szerinti találmány hasznosítására."*

E körben természetesen alapvető fontosságú a „számottevő gazdasági jelentőségű műszaki előrelépés" fogalmának tisztázása, illetve a kényszerengedélyek esetén – az alap szabadalomra épülő technológia viszontjogosításának feltételei.

A kényszerengedélyezés általános szabályai országonként eltérnek. Azonban – ahogy az Európai Unió korábban már rendeletbe foglalta a közegészségügyi problémákkal küzdő országokba történő kivitelre szánt gyógyszeripari termékek előállításával kapcsolatos szabadalmak kényszerengedélyezésének szabályait[39] – elképzelhető lenne egy – a technológiai fejlődést segítő kényszerengedélyezési rendszer kidolgozása. Talán épp ez a több tagállamot érintő háborúskodás elvezethet egy ilyen békekötési paktumhoz. Az mindenesetre biztos, hogy a Bizottság az üggyel foglalkozni kíván, ahogy ezt egy sajtóközleményben jelezte is[40].

Itt kell azonban megjegyezni, hogy a szabadalmi kényszerengedélyek a formatervezési mintaoltalmakat nem érintik, hiszen a design nem kötelező elem, arra vonatkozóan a korábban már említett korlátozások vonatkoznak.

---

[38] Erről részletesebben itt:
`http://fosspatents.blogspot.com/2011/07/apple-says-samsung-has-abusively.html`

[39] Az Európai Parlament és a Tanács 816/2006/EK Rendelete:
`http://eur-lex.europa.eu/LexUriServ/site/hu/oj/2006/l_157/l_15720060609hu00010007.pdf`

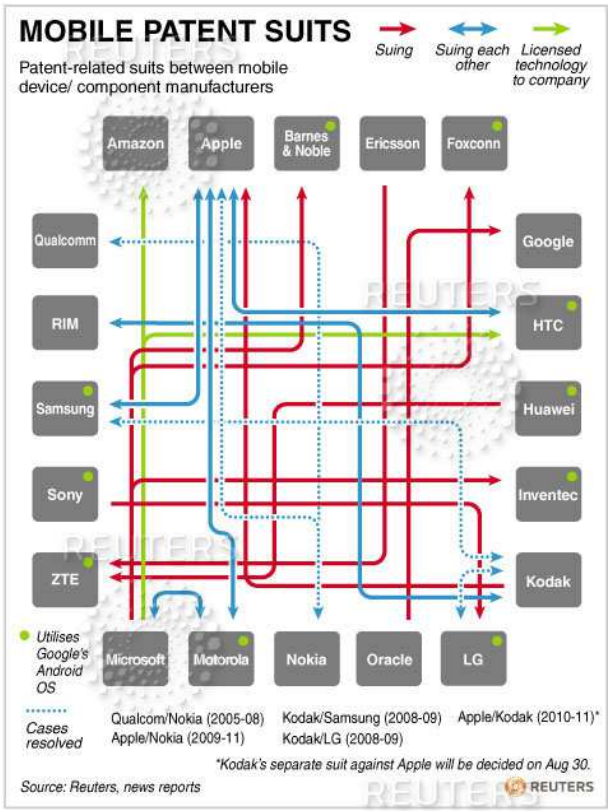
[40] *„The Commission has indeed sent requests for information to Apple and Samsung concerning the enforcement of standards-essential patents in the mobile telephony sector. Such requests for information are standard procedure in antitrust investigations to allow the Commission to establish the relevant facts in a case. We have no other comments at this stage"*

5. Diplomáciai hadviselés

Feltétlenül érdemes kitérni arra, hogy a háborút az Apple nem csak a Samsung ellen vívja, hanem csatában áll még a HTC-vel és a Motorolával (a kivásárlást követően ismertebb nevén: Google), valamint a Nokiával is[41].

Természetesen a hadiállapot ennél jóval több, a mobiltelefónia és tabletek piacán érintett fél között is fennáll, mindenkinél különféle okokból. Ami azonban jól látható, hogy létezik egy androidos tábor (amit természetesen a Google erősít), és ők így egyben az Apple legnagyobb ellenfelei.

A szövetségesek és ellenfelek felállása valami hasonló:



Hogy jön ide a diplomácia, tehetnénk fel a kérdést. Természetesen fehér kesztyűben... A Samsung legnagyobb megrendelője ugyanis továbbra is az Apple, ahogy az Apple részére az iparjogvédelmi eljárásokban érintett valamennyi készülék tartalmaz a Samsung által gyártott elemeket (pl. processzor, memória modul, lcd). Az eddigi eljárások alapján többek között az Apple jogdíjat fizet a Nokiának, a HTC és a Samsung a Mircosoftnak.

6. Hidegháború

Elnézve hát a csatatereket, a hadvezéreket és a fegyvereket, megállapíthatjuk, hogy bizony egy hosszabb háború vette most kezdetét. A frontvonalak folyamatosan átrendeződnek, az iparjogvédelmi arzenál mellett felbukkannak versenyjogi elemek. Egy olyan háborúnak vagyunk tanúi, amely-

[41] A grafikon származási helye: `http://graphics.thomsonreuters.com/RNGS/2011/AUG/PATENT_CI.jpg`

ről jogi egyetemre járó unokáink majd mint a híres „Apple – Samsung jogvita” fognak megemlékezni, és doktori értekezések és memoárok születnek majd mindkét oldalon… Egy azonban biztos: a fejlődés nem áll meg, mert most jött el az okostelefonok és tabletek kora.

XenUI – Xen User Interface

Hargitai Zsolt

Kivonat

Napjaink egyik kikerülhetetlen trendje a virtualizáció. A vállalatok és szervezetek, az egészen kicsiktől a nagyokig igyekeznek optimalizálni a rendelkezésre álló erőforrások kihasználtságát. Az egyik egyszerűen megérthető és gyors megtérüléssel kecsegtető terület a szerver hardverek számának minimalizálása virtualizáció alkalmazásával.

Tartalomjegyzék

1. A virtualizáció területe **64**

2. Nyílt forráskódú technológiák **64**

3. A XenUI termékről **64**
3.1. Legfontosabb funkciók 65
3.2. Tervezett fejlesztések 65

1. A virtualizáció területe

Amikor egy informatikai szakember a virtualizáció irányába akar elindulni, több út is kínálkozik számára. Több neves gyártó kínálja megoldásait:

- VMware vSphere
- Microsoft Hyper-V
- Xen és KVM technológiák

Mindegyiknek megvannak a maga előnyei és hátrányai, amíg a technikai oldalt nézzük; elismerve, hogy a piacvezető VMware, menedzsment szolgáltatásait tekintve, jelenleg társai előtt jár. A technikai szempontokon kívül azonban még egy fontos tényezővel kell számolnunk, amely éppen az eredeti költségcsökkentési célunkat akadályozhatja: ez pedig a beruházásra fordítandó összeg. A gyártóközi megállapodásoknak vagy termékcsatolásnak köszönhetően néha ingyenesen juthatunk hozzá ezekhez a megoldásokhoz. A kivételektől eltekintve azonban általános érvényű szabály, hogy a fizikai gépek mennyiségének (vagy a bennük található processzorok/magok számának) növekedésével arányosan egyre több pénzt kell áldoznunk egy virtualizációs infrastruktúra megvalósítására és fenntartására. (Mellékesen a fenti „ingyenes” megoldások esetén is szükségesek további fizetős komponensek egy jól működő architektúra kialakításához.)

2. Nyílt forráskódú technológiák

Azok számára, akik a fizetős megoldások helyett a nyílt forráskódú motorok irányába döntenek, alapvetően két megoldás kínálkozik: a KVM és a Xen. Mindkettő támogat Windows és Linux vendég operációs rendszereket (ezen felül még számtalan egyéb OS-t is), mindkettőnek megvannak a maga kvantitatív előnyei a másikkal szemben (CPU és memória-kihasználtság vs. írási és olvasási sebesség), és szolgáltatásban, valamint támogatásban is hasonlóan erős háttérrel rendelkeznek, hiszen míg a KVM mögött a RedHat, addig a Xen mögött a Citrix és a Novell vonultat fel fejlesztői kapacitást és támogatást. A döntés tehát sok esetben egyedi érzelmi kötődésen vagy az elérhető tudáson múlik. A Novell hazai fejlesztőközpontja a Xen mellett tette le voksát, főleg mivel az nagy méretű szerverfarmok esetén jobb teljesítménymutatókat tudott elérni, mint vetélytársa, ezzel hatékonyabb kihasználtságot biztosított.

3. A XenUI termékről

Az elmúlt évek magyarországi virtualizációs projektjei során a Novell hazai fejlesztői az ügyfelek igényei alapján kezdtek el kidolgozni menedzsment felületet a Xen alapú virtuális gépek felügyeletére. Az eleinte még csak egy- két alapszolgáltatást biztosító fejlesztés mára XenUI néven külön termékké fejlődött. A termékfejlesztés fő célja, hogy elérhető árú, webes interfésszel rendelkező, nagy méretekben skálázható virtualizációs menedzsmentrendszert készítsünk a Xen motorhoz, főleg azokra az alapvető funkciókra koncentrálva, amelyeket a hazai nagyméretű szerverfarmok üzemeltetői hasznosnak találnak. Azért szükséges külön kiemelni a „hazai” szót, mert bár itthon ezek a farmok a legnagyobbak közé tartoznak, nemzetközi viszonylatban csak a középmezőnybe sorolódnak. Éppen ez adja a XenUI létjogosultságát, hiszen a gyártók által a hazai nagyméretű ügyfeleknek ajánlott megoldások gyakran feleslegesen sok és kihasználatlan funkciót tartalmaznak, amelyeknek fejlesztési és marketingköltségét ugyanúgy meg kell fizetni a termék árába építve.

3.1. Legfontosabb funkciók

Tekintsük át röviden a termék legfontosabb szolgáltatásait:

- Vendég gépek létesítése, megsemmisítése, indítása és leállítása
- Tetszőleges számú vendég és gazda operációs rendszer kezelése
- Windows és Linux vendég operációs rendszerek párhuzamos kezelése
- CPU, memória és HDD információk a vendég gépekről
- Egyszerűbb riportok és grafikonok a hisztorikus futtatási adatokról
- Jogosultságok és szerepek definiálása, valamint kiosztása

3.2. Tervezett fejlesztések

A termék fejlesztése jelenleg is folyik, és a közeljövőben az alábbi funkciók megvalósítása várható:

- Vendég gépek migrálása gazda rendszerek között
- Vendég gépek hardver elemeinek (CPU, memória, HDD) futtatás közbeni változtatása
- Vendég gépek hálózati kártyáinak futtatás közbeni konfigurációja
- Mobil felügyeleti alkalmazás fejlesztése
- Vendég gépek csoportos kezelése
- Csoportos jogosultságkezelés

Áttörni a falat
(Szabad szoftverek egy önkormányzatban)

Karay Tivadar

Kivonat

Budapest XVIII. kerülete, Pestszentlőrinc – Pestszentimre, Budapest déli széle, peremkerület. Nincs informatikai szempontból semmi különlegesség, nincs itt Infopark, nincs egyetem, nincs jelentős idegenforgalom. A sok lakos, a telkek, kertes házak, építkezések, no meg a szociális segélyek itt is, mint mindenütt, jelentős munkát adnak a Polgármesteri Hivatalnak. Kollégáink, az ügyintézők és hivatali alkalmazottak sem informatikai doktorátussal választották a köztisztviselői hivatást. Egy átlagos hivatal vagyunk, de ebben az átlagos környezetben nem átlagos az, hogy megpróbáljuk a „szokásos” informatika falát áttörni, és a szabad szoftvereket minél nagyobb arányban használni.

Tartalomjegyzék

1. Mekkora a fal? (Mik az akadályok?) 68
1.1. Az államigazgatási informatika jellegzetességei . . . 68
1.2. Kereskedelmi szoftverek . . . 69

2. Ki áll a fal elé? (Felhasználók és egyebek) 69

3. Van-e kalapácsunk? (Milyen eszközeink vannak?) 70
3.1. Vállalkozó segítségével . . . 70
3.2. Oktatás . . . 71

4. Lyukak a falon (Amit elértünk.) 71
4.1. Linux kliensek . . . 71
4.2. OpenOffice.org/Libreoffice . . . 72
4.3. Stratégia . . . 72

5. Miért van fal? (Mit tehetnek a magasabb szervek?) 73
5.1. Ajánlások, szakmai segítség . . . 73
5.2. A jogalkotás lehetőségei . . . 74

6. Élet fal nélkül (Reménykedjünk. . .) 74

1. Mekkora a fal? (Mik az akadályok?)

1.1. Az államigazgatási informatika jellegzetességei

Az államigazgatásiban az elvek, módszerek és az ezekhez kapcsolódó gyakorlatok nagyon lassan változnak. Mondhatnánk, II. József törvényei is meglátszanak a mai szokásokon. Ehhez a tempóhoz képest berobbant a mindennapokba az informatika, és kezdett egyre több munkafolyamatban szerepet játszani. Ez a számítógéppel segített papír alapú ügyintézés korszaka, amely nagyon sok területen még ma is tart. Mit takar ez?

- Az állampolgárok jellemzően papíron adják be kérvényüket, terjesztik a hivatal elé problémáikat.
- A hivatal papír alapon állít elő végzést,vagy határozatot.
- Az ügyek elbírálásához tartozó adatok, iratok (pl. társszervek igazolásai stb.) jellemzően szintén papíron érkeznek meg.
- Az ügyek tárolása így alapvetően papíron, dossziékban történik.
- Az informatika egyes (bár nagyszámú) mellékadatokat kezel (költségvetési, szociálissegély-adatok, térinformatika), amelyek végeredménye azonban papíron jelenik meg.
- A számítógépet jórészt írógépnek tekintik, a töredékét használják ki a benne rejlő lehetőségeknek.
- Sok kisebb-nagyobb, egymástól független szoftvert használunk.
- Az állampolgárokra vonatkozó adatok egységes adatbázisban történő tárolása nem valósult meg, jórészt törvényi akadályok miatt sem.

Pedig az informatika sok területen más munkamódszert, más ügyvitelszervezést kívánna meg, illetve tesz lehetővé. A „számítógéppel segített” ügyintézés olyan, mintha a gépkocsit arra használnánk fel, hogy húzzuk vele a szekeret. Nyilván több tekintetben előnyösebb, mint a ló, de maga az egész rendszer valahogy nem az igazi.

Mitől alakult ez így?

A papír alapú iratkezelés nagyon régi kialakult gyakorlattal rendelkezik. A számítástechnika először egy-egy PC formájában jelent meg, segítő funkciókat látott el, nem volt se igény, se lehetőség arra, hogy az államigazgatási módszereket és gyakorlatot ne a papír alapú, hanem az informatika alapú lehetőségekhez kössük. A 2004. évi, azóta módosított „A közigazgatási hatósági eljárás és szolgáltatás általános szabályairól” szóló törvény (KET) volt az első, amely egyes elemeiben megpróbált informatikai alapon új közigazgatási módszereket bevezetni. Ez a törvény jelentősen módosult, még mielőtt a szemléletet megváltoztatta volna. Az informatika fejlődését az államigazgatásban jelentősen hátráltatta az elektronikus aláírás kontra ügyfélkapu háború, ami gyakorlatilag azt eredményezte, hogy az elektronikus aláírás rendszere csak nyomokban létezik, az ügyfélkapu pedig lassan fejlődött.

Az államigazgatási informatika tekintetében az egyik ideális megoldás a hivatalok teljes működését lefedő integrált szoftver lehetne, ez azonban csak nagyon kevés helyen valósult meg, és még az eszméje sem terjed igazán.

1.2. Kereskedelmi szoftverek

Jól tudjuk, hogy az „állami pénz” laza elköltése mindig nagy kísértés az államigazgatás döntéshozóinak. Az önkormányzatok és az államigazgatás felelős vezetői számára az informatika (épp a fentebb említett történelmi okok miatt) nem húzóágazat, hanem csak egy eszköz. Az eszközzel való törődést könnyű úgy gyorsan elintézni, hogy a rengeteg, hardvert és szoftvert kínáló vállalkozót megbízzuk egy-egy projekt elintézésével. „Le van a gond.” Így aztán kialakult a rendkívül heterogén gép- és szoftverpark, fizetős szoftverekkel, amely szoftverek között kis, egyszemélyes cégek által gyártott célszoftverektől a közismert óriás alkalmazásaiig minden megtalálható. Az önkormányzati célalkalmazások kizárólag kereskedelmi szoftverek, mivel alkotójuk szeretne hasznot realizálni rajta, az általános programok meg a „legelterjedtebb” kínálatából kerülnek elő.

Ezek azok a tények, amelyek jelentős falat alkotnak az államigazgatásban a szabad szoftverek terjedése elé, és amely falat az informatikusok kötelességből nem, legfeljebb csak megszállottságból (mazochizmusból) próbálnak áttörni.

2. Ki áll a fal elé? (Felhasználók és egyebek)

Természetesen ilyenkor illenék jól leszidni a szerencsétlen felhasználókat, és kicsit óvatosabban, de távollétükben jól leszedni a keresztvizet a főnökökről is.

Értsük meg őket, akkor sokkal könnyebben megy.

Egy ügyintéző azt szeretné, ha mindennapi munkáját szép sorban, rutinból, feleslegesnek ítélt változások nélkül tudná végrehajtani. Épp elég baj neki a sok, nem mindig kulturált ügyfél, az állandóan (Magyarországon jellemzően sokszor) változó jogszabályi környezet, a négy évente, választásonként bekövetkező változások. Púp a hátára, hogy ehhez képest a számára érdektelen informatika túl gyorsan változik. „Miért nem lehet a régi jó Windowst használni” - kérdezik, de mi tudjuk, hogy igazából a 2000-es Word szövegszerkesztőre gondol, amit nem szeretne lecserélni. „Nem írok körlevelet, nem vagyok én informatikus.” „Igaz, nem mentettem el a levelet, de megírtam, ott lesz a gépen, találjátok meg!” Ilyen és ehhez hasonló mondatokat gyakran hallanak a hivatalok informatikusai.

Már azt is nehéz nekik megmagyarázni, hogy miért változnak időnként az alkalmazások. Miért kötődik az (elromlott) géphez a régi szoftver, ha OEM licenceléssel lett vásárolva? Vagy egész egyszerűen, miért van másutt egy ikon, mint ahol eddig megszoktam?

Bármennyire is furcsák informatikus szemmel ezek a felvetések, mégsem várhatjuk, hogy a hivatalnokok megértsék egy számukra idegen szakterület sajátosságait.

Többször tapasztaltam, hogy egy, az ügyintéző munkáját könnyítő szoftverváltozást sem éltek meg fejlődésként, mert az átállás nehézségei, a betanítás-betanulás félelmei, a megváltozott munkamódszerekből adódó gyakorlatlanság elfeledtették a felhasználókkal, hogy tulajdonképpen pozitív változás történt.

Természetes gyanakvás is van sok emberben. Miért akarjátok ezt a szoftvert? Miért pont nekem? Miért pont most? Egyszer – még korábbi munkahelyemen – MS Word helyett MS Worksot akartam bevezetni, ami jóval olcsóbb volt, és a feladatnak bőven megfelelt. Meggyanúsítottak, hogy nyilván rokonom a programozó...

Lépten-nyomon előfordul, hogy egy szabad szoftver bevezetésénél (amely 99%-ig tudja mindazt, amit a korábbi), a felhasználó felismeri a hiányzó egy funkcionalitást (ami a munka szempontjából nem fontos, és amit egyedül ő használ), és közli hogy „Az *egész* rossz.” (Persze nem egészen ilyen szalonképesen.)

Az informatikus ebben a helyzetben köteles alkalmazkodni a *valós* igényekhez, és finoman elterelni a figyelmet az értelmetlen igényekről, vagy a céltalan hőzöngésről. A logikus határig ki kell szolgálni az alkalmazókat, de meg kell tanítani nekik, hogy ez a szakma állandó változással jár, és ez nem rajtunk múlik.

Könnyebb a helyzet, ha a hivatal vezetői pártolják a munkánkat. Általában minden vezető örül a költségcsökkentésnek, amit a szabad szoftverek lehetővé tesznek. De uralkodik az a szemlélet is hogy „Csináljatok bármit, csak ne változzon semmi.”, vagy „Nyugodtan cseréljétek le a szoftvereket, csak *az enyémet ne*”. Ez utóbbi magatartás tud a legnagyobb gátjává lenni a fejlődésnek. Ha egy főnök belátja, hogy a változás szükséges és hasznos, akkor ne próbáljon a helyzetét kihasználva háttérben maradni. A harctéren is jobban hat az „Utánam!” vezényszó, mint az „Előre!”.

Bármily meglepő, a nem szabad szoftverek gyártói nem állnak a fal elé. Ez is bizonyítja, hogy itt nem harc van, hanem egy értelmes feladat- és piacmegosztás. Sőt, egyre többen jelentkeznek olyan cégek, amelyek szabad szoftverekkel kapcsolatos szolgáltatásaikat ajánlják fel.

3. Van-e kalapácsunk? (Milyen eszközeink vannak?)

Az államigazgatás szempontjából szóba jöhető szabad szoftverek jók. A legtöbb helyen, ahol szabad szoftverrel kiváltunk egy kereskedelmit, nem csökken sem a minőség, sem a rendelkezésre állás, sem a biztonság. (Nem akarok belemenni abba az örökké dúló vitába, hogy a szabad szoftver, vagy a zárt a biztonságosabb, a jelenlévők nyilván elkötelezettek, a másik oldalon állók meg érdekeltek.)

A szabad szoftvereket tehát nem kell megvédeni, megvédik ők magukat. Ezen a fórumon nem kell elmondani azokat az előnyöket, amelyeket mindenki tud a gyártófüggetlenségtől, a sok tesztelő által nyújtott biztonságon keresztül a változtathatóságig.

Az államigazgatásban és az önkormányzati rendszerben fontos előnyöket emelném ki. A költségek csökkentése mindig is fontos szempont volt, most válságos időszakban különösen. Ez akkor is igaz, ha egyesek kimutatni vélik, hogy az átállás nehézségei, az alkalmazott partnernek fizetett megbízási díj csökkentik a nominális hasznot.

3.1. Vállalkozó segítségével

Vegyük azt az esetet, hogy egy szabad szoftverre való átálláshoz külső vállalkozót veszünk igénybe. Gondoljunk bele: ha az országot elhagyó licencdíj helyett akár ugyanannyit is fizetünk egy helyi vállalkozónak, már az is haszon. A vállalkozó partnerré válik, belelát a munkánkba, alkalmazkodni tud a hivatali ügyintézés sajátosságaihoz, érdemben tud segíteni. A vállalkozó is tapasztalatot szerez, adott esetben képessé válik új, csak számunkra fontos megoldások, kiegészítések elkészítésére. A partner itt él velünk, egy országban, akár egy városban, akár potenciális ügyfél is lehet, véleménye, elképzelései állampolgári igényként számunkra is fontosak lehetnek (Arról nem beszélve, hogy a neki fizetett díj itthon adózik.).

Ha másnak nem, az államigazgatásnak kellene élen járnia abban, hogy az ebben a körben felmerülő problémákat hazai megoldásokkal oldjuk meg. Ha büszkék vagyunk a szabolcsi almára, a tokaji borra, és mozgalmak vannak a magyar áru védelmére, akkor az államigazgatásnak illene példát mutatnia a magyar programozói tudás hasznosításában is.

Másrészt nem kell minden szabad szoftverre történő átálláshoz partnert igénybe venni. A szabad szoftverek nagy előnye a dokumentumok könnyű elérhetősége, a nyilvánosság, de legalább is rengeteg kolléga, szakember, támogató tud segítséget adni a hivatal informatikusainak. Ha egy új szoftver alkalmazását nem készen kapjuk, hanem magunk kísérletezzük ki, akkor lesz csak igazán a miénk, akkor tudjuk átlátni és kihasználni tulajdonságait, előnyeit.

3.2. Oktatás

A legfontosabb eszközünk az oktatás lehet, lehetne. A szabad szoftverek alkalmazásán kívül is az ügyintézők tudását időről-időre fel kell, kellene frissíteni. A törvényi változások, szoftver-verziócserék mind-mind módosítják a munkamódszereket, és az egykor jó szokások erodálódnak, vagy egyenesen hátrányossá válnak. Az ügyintéző nyilván hajlamos a számára legkönnyebb munkamódszert kialakítani, ez azonban a hivatal egésze szempontjából káros lehet. Ha a saját dokumentumait rendszeresen a saját gépére menti a kolléga, az számára kézenfekvő, de így más nem hasznosíthatja, továbbá egy merevlemezhiba esetén az egész eltűnik.

Az oktatás így egyben minőségbiztosítási szerepet is ellát amellett, hogy nem egy pillanatnyi hiba okozta ideges állapotban, hanem nyugodt, tantermi körülmények között lehet elmondani a kollégának, mit tud az új szoftver, miért van rá szükség, miért ezt választottuk, hogy kell használni. Lehet, hogy oktatóként „hazabeszélek”, de minden oktatással, meggyőzéssel „elvesztegetett” munkaóra később többszörösen hasznosul a mindennapi munkában.

4. Lyukak a falon (Amit elértünk.)

Nem a mi hivatalunk érte el a legjobb eredményt a szabad szoftverek alkalmazásában. Jelen szekció előadói között is ott vannak példaképeink, akiktől tanulunk, és szeretnénk tovább is tanulni. De azért elértünk valamit.

15 szerverünkből 2 az, amin nem Linux fut, egyiken Novell fájlszerver (amit azóta ugyancsak lehetne Linuxon futtatni), a másik egy Windows 2000 szerver néhány régi alkalmazás kedvéért.

Hogy kedvet adjak a Linux szerverekhez: A Novell NetWare rendszert manapság eleve Linuxra telepítik. Az Oracle adatbázis-kezelő, és minden más Oracle szoftver, beleértve a webszervert, alkalmazáskiszolgálót, Application Expresst, remekül elfut Linuxon. Tűzfal, levelezőszerver általában az első, amit költségtakarékossági okokból mások is Linuxra telepítenek, és ezek nyilván szabad szoftverek. A belső Apache alapú weblap, a JBoss alkalmazásszerver, a SquirrelMail webes levelező szabad szoftverek, és Linuxon futnak. A belső hibafigyelés Nagios szerver alkalmazásával történik.

Ez persze nem nagy eredmény, sok más államigazgatási szervnél előfordul, hogy adatbázisszervert, webszervert, levelezőszervert stb. Linux szervereken futtatnak.

4.1. Linux kliensek

Mi pár éve elkezdtünk Linux klienseket is telepíteni. 270 felhasználói gépből jelenleg több, mint harmincon valamilyen Linux disztribúció fut. Több gépen is lehetne, de amíg egy régi gép fut a rajta levő OEM szoftverekkel, addig nem cseréljük le. A linuxos kliensek beilleszkedtek a hivatal életébe. Sikerült megoldani néhány, az államigazgatásban kötelező szoftver futtatását a Wine emulátorral. A platformfüggetlen alkalmazások, köztük a hivatal számára legfontosabb iktató szoftver nyilvánvalóan működik Linuxon. A hibajavításhoz és felhasználótámogatáshoz szükséges távoli elérés is megvalósult. Megoldottuk, hogy előretelepített gépből klónozással „más operációs rendszerekhez képest” sokkal gyorsabban tudunk Linuxot telepíteni, a szükséges alkalmazásokkal, a hivatalra és az informatikai rendszerre jellemző beállításokkal együtt.

Természetesen ez a munka nem ment zökkenők nélkül. Máig sem értjük, hogy a hivatalban igen fontos Complex jogtár linuxos változatának fejlesztése miért állt le, pedig jól működő megoldás volt. Így Wine-nal kell üzemeltetnünk.

Az önkormányzatok számára fontos és nehezen kikerülhető DOS-os program (ÖNKADÓ) jól működik emuláció alatt. Azonban a róla történő nyomtatás szerver vezérelte hálózati nyomtatóra sokáig nem működött. Egy kollégám hónapokig kísérletezett vele, nyilván csak maradék időben. Sikerült, aztán külön gond lett az ékezetes karakterek helyes kezelése, végül az is megoldódott. Előtte egy cég ajánlatot tett a megoldásra, de akkora összegért, amit nem tudtunk kifizetni. Nem tudom, hogy ez arra jó példa-e, hogy külső segítség nélkül is tudunk boldogulni, vagy arra, hogy pénzért sok munkát és időt lehetett volna megtakarítani.

4.2. OpenOffice.org/Libreoffice

Az OpenOffice.org/Libreoffice-ra való átállás sem zökkenőmentes. A pletykákkal ellentétben nem egyes hiányosságok okozzák a zavart, hanem az a rossz gyakorlat, hogy a kollégák úgy szeretnek dokumentumot létrehozni, hogy egy korábbi, hasonló tárgyú dokumentumot átírnak. Képzeljünk el egy dokumentumot, amit Word 95-ben kezdtek megírni (szépen szóközzel középre húzva a megszólítást), aztán különböző felkészültségű ügyintézők minden lehetséges Word verzióval belejavítottak (van aki ismeri a stílusokat, a többség nem), és a végén ezt OpenOffice.org Writerrel akarjuk megnyitni, és doc fájlként elmenteni.

Belátható, hogy az irat tulajdonképpen semmilyen szövegszerkesztőn nem mutat jól, a papíron talán nem látszik, de hemzseg a belső hibáktól. Ezt egy SZABVÁNYOS eszközzel megnyitva sok hiba a napvilágra jön.

Persze lehet kidolgozni informatikai megoldásokat a kompatibilitási hibák csökkentésére, de ez tipikus példa arra, hogy egy eleve rossz iratkezelési módszert nem lehet jól támogatni számítástechnikai eszközökkel. Nem csak informatikai, hanem iratkezelési, adatbiztonsági szempontból is jobb módszer, ha a hivatal előre elkészített sablonokat használ az iratok készítéséhez. Ez nyilván nem informatikai átszervezést igényel, de az informatika könnyedén tudja támogatni, lényegesen kevesebb ráfordítással, mint a rossz gyakorlat fenntartását.

4.3. Stratégia

A technikai faltörésünkön kívül büszkék vagyunk a stratégiánkra. Pár éve kialakítottuk és sikeresen megvalósítjuk azt az elvet, hogy ha a számtalan különféle szoftverből összeálló rendszert nem is tudjuk egységesen lecserélni, új szoftvernek lehetőleg platformfüggetlent, még inkább webes vagy JAVA WebStartos kialakításút vegyünk. Szívesen mondanám, hogy alkalmazzunk szabad szoftvert, de önkormányzati célalkalmazások (adó, szociális ellátás, ingatlan- nyilvántartás stb.) piacán nem jellemzők a szabad szoftverek. Jó példaként mondom: az elmúlt 3 hónapban két olyan ajánlatot is kaptunk cégektől, amely szabad szoftverre alapuló megoldást vezetne be.

Ez a stratégia hosszú távon biztosítja a mozgásterünket a Linux és bármely más várható informatikai megoldás irányába. A webes megoldások külön előnye a távoli hozzáférés biztosítása, az otthoni távmunka megoldásának lehetősége. Ez is egy olyan terület, amely várhatóan nemzetgazdasági megtakarítást hozhat. Talán nem durva kijelenteni, hogy ha egy államigazgatási alkalmazás nem webes technológiára épül, az egy kicsit le van maradva, és akkor talán más, szakmai területen sem élvonalbeli.

Mellékesen 2009-ben elkötelezettségünk bizonyítására, no meg hogy tanuljunk az okosabbaktól, szakmai konferenciát rendeztünk „Szabad szoftver az Önkormányzatokban” címmel. Azóta egy honlapot is üzemeltetünk ebben a témában: `http://szabadszoftver.bp18.hu/`. Ez itt a reklám helye: olvassátok, és írjatok bele.

Itt szeretném név szerint is megemlíteni kollégáimat, Szállási Zoltán csoportvezetőt, Koppányi Ferenc, Erdős Attila és Prazsák Péter informatikusokat, hiszen a fal áttörése csapatmunka, amelyben mindenkinek szerepe van.

5. Miért van fal? (Mit tehetnek a magasabb szervek?)

Ebben a körben nyilván köztudott, hogy az államigazgatás, és az önkormányzati rendszer átalakulóban van. Még nem igazán tudjuk, hogy az átalakulás mire vezet, végképp nem, hogy milyen informatikai változást hoz magával. Reméljük, hogy az a világot átfogó mozgalom, amit a szabad szoftverek alkalmazása teremt, a magyar közigazgatásba is behatol. Örömmel nyugtázhatjuk, hogy az elmúlt időkben konferenciákon újra és újra szóba kerül a szabad szoftverek közigazgatási használatának ügye, most is örülünk a kormányzat részéről jött előadóknak. Hallani lehetett az ODF formátum államigazgatási bevezetéséről is – szívesen látnánk ezt megvalósulni.

Mit lehetne még tenni?

5.1. Ajánlások, szakmai segítség

Bármilyen sokan is gyűltünk itt össze, sosem képviselünk akkora marketing erőt, mint amit a jogdíjakból tudnak a profitorientált vállalkozások megteremteni. Az állampolgárok, az önkormányzati vezetők, hivatali döntéshozók, ha nem informatikusok, nyilván jobban el vannak árasztva reklámmal, mint a miáltalunk küldött információkkal. Amennyiben a közigazgatás vezetése egyensúlyi helyzetet szeretne teremteni, érdemes lenne ajánlásokat közzétenni, a jó megoldásokat publikálni, példát nyújtani.

A közigazgatást gyakran szidják, hogy pazarló. Itt az alkalom jó példát felmutatni! Egyszer soksok évvel ezelőtt felmerült, hogy ha a kormányzat egy szabad szoftver megírását támogatja, akkor az minden magyar hivatal és önkormányzat számára ingyen elérhető lehet, ahelyett hogy mindenki külön-külön megvásárolná. Egységesítést, interoperabilitást is sokkal könnyebben lehet így elérni.

Természetesen igazi nyílt forráskódú szabad szoftverre gondolok, nem arra, hogy valamely hatóság egy szoftvert kiválaszt, és kötelezővé tesz. Ez olyan versenyhiányt eredményez, ami meglátszik azon, hogy a közigazgatásban még floppyról működő DOS-os szoftverek nem kis számban használatosak. Sajnos ismét meg kell említsem az önkormányzati adózási rendszert, aminek a DOS-on túlmutató verziója több mint 11 éve készül, és amit láttunk belőle, félkészen, az már most elavult.

Egy nyílt forráskódú szoftver önmagában hordozza a versenyt. Ha az eredeti fejlesztő nem elég naprakész, majd lesz más, aki kiegészíti. A sok-sok magyar hivatal informatikusai is potenciális programozók lehetnek: mindenki a maga területének megfelelő tudást tudja hozzátenni egy nyílt alkotáshoz. És ez még olcsó is.

Nagy álomnak tűnik ez, de mi akik itt vagyunk, tudjuk, hogy ez működik, és remek alkalmazásokat eredményez.

Magam részéről: minden hivatali, oktatási és otthoni informatikai igényemet ki tudja elégíteni a szabad szoftverek halmaza. Ha felmerül egy igény (legutóbb CAD programra volt szükségem), kis internetes keresés után már többet is találtam, és már csak válogatni kellett.

Régen volt egy informatikai tárcaközi bizottság. Egy ilyen, vagy hasonló szerv megtehetné ezt a feladatot: különböző közigazgatási feladatokra összegyűjti a megoldásokat, esetleg ajánlásokat dolgozna ki, még jobb esetben támogatná az ígéretes kezdeményezéseket. Ezzel segíthetnék a különböző hivatalok, önkormányzatok döntéshozóit.

5.2. A jogalkotás lehetőségei

Még egy meglátásom van, amiért a jogászok meg fognak égetni. Nyilván nem egy számítástechnikai megoldást alapul véve kell jogszabályokat hozni, de nem ártana kodifikálás közben a megoldhatóságra odafigyelni. A paragrafusok alkotása közben gondolni kellene arra, hogy az ügyek folyamatokban testesülnek meg, a folyamatokat a szoftverek jól tudják modellezni, míg a jogrendszer nem folyamatorientált. Nem kellene annyi redundáns adat, nem kellene annyi független azonosító, nem jó, ha különböző hivatalok más és más adatokat tárolnak ugyanarról az objektumról.

Nyilván a betonút az, ami adott, amihez tervezik az autót. De az út tervezésekor is érdemes figyelembe venni, hogy milyen gépkocsik vannak forgalomban.

Ennek a hasonlatnak megfelelően a jogalkotásnál, munkamódszerek megalkotásánál, űrlapok és nyilvántartások előírásánál jó lenne az informatikai megvalósíthatóságra gondolni.

6. Élet fal nélkül (Reménykedjünk. . .)

Messzire vezetne, ha elmondanám az álmaimat a közigazgatási informatikáról. Nyilván részben mint informatikus, részben mint állampolgár tenném meg. Jó lenne egy egészséges arány a szabad- és a kereskedelmi szoftverek között, jó lenne több webes, sőt mobiltelefonos szolgáltatás, jó lenne egy olcsóbb közigazgatás, jó lenne a kismamákat, részmunkaidősöket alkalmazó távmunkás ügyintézés.

Nem kell nagyon aggódni, hogy ez a folyamat lassan megy. Stallmann 1983-an alapította a GNU-t, ez a közel 30 év csak egy röpke pillanat a magyar közigazgatás időszámításában, összevetve például azzal, hogy jelenleg több DOS-alapú alkalmazás használata kötelező. De ez nem gúnyolódás akar lenni: az is nagyon nagy baj lenne, ha a közigazga-

tási szoftverek, illetve a rendszerek iránti igények olyan gyorsan változnának, mint a szórakoztató elektronika. Elég baj az, hogy a jogszabályok túl gyorsan változnak, de ez nem a mi asztalunk.

Befejezésül egy divatos technológiáról: Jó lesz-e, ha fal helyett felhő lesz? Amennyiben a múlt megismétlődik, és a cloud-technológia ismét néhány külföldi nagyvállalat uralmát jelenti nemcsak a hardverek és szoftverek, hanem immár az adataink fölött is, akkor *nem*.

Amennyiben egységes, átlátható ügyintézést, platformfüggetlen kiszolgálást, minimális költségeket eredményez az adataink maximális állami- és civil kontrollja mellett, mondjuk egy önálló magyar közigazgatási felhőben, akkor igen. De ehhez még meg kell harcolni a mi harcunkat.

Egy biztos: a fal nem szép.

Android – két évvel később

Kis Gergely
<gergely.kis@mattakis.com>

Tartalomjegyzék

1. Bevezetés **78**

2. Az Android története **78**
2.1. A kezdetek 78
2.2. Megjelenik az Android 1.0 és a T-Mobile G1 79
2.3. 2009: 4 kiadás egy év alatt 1.1, 1.5, 1.6, 2.0 79
2.4. 2010: Az okostelefon piac meghódítása 79
2.5. 2011: Tabletek és Jégkrémszendvics 79

3. Az Android 4.0 (Ice Cream Sandwich) újdonságai **80**
3.1. Felhasználók 80
3.2. Alkalmazásfejlesztők 80
3.3. Platformfejlesztők 81

4. Szemelvények az Android platform belsejéből **81**
4.1. Az Android Open Source Project – első lépések 81
4.2. Folyamatok, szolgáltatások és ami mindent összeköt: Binder 84
4.3. Az Android portolása új hardverre 86
4.4. Hová tűnt a forráskód: a kernel.org incidens 87

5. A sötét oldal **87**
5.1. Elég nyílt-e az Android? 87
5.2. Elárvult készülékek, a platform fragmentációja 89

6. Kitekintés a jövőbe **89**

7. A szerzőről **90**

1. Bevezetés

Az elmúlt bő három évben az Android gyakorlatilag a semmiből nőtte ki magát az okostelefon piac egyik meghatározó szereplőjévé. A legutóbbi Szabad Szoftver Konferencián tartott előadásom óta sok minden történt az Android világban, megjelentek a tabletek, és immár az Android 4.0 is a megjelenés küszöbén áll.

A továbbiakban áttekintjük a platform történetét valamint jelenlegi helyzetét. Megvizsgáljuk az erősségeit, de nem siklunk el a gyengeségei felett sem. Bemerülünk a felszín alá, és felfedezzük az Android-világ néhány kevésbé ismert, érdekesebb részletét. Végül kitekintünk a jövőbe, a spekulációk és összeesküvés-elméletek világába.

2. Az Android története

2.1. A kezdetek

Kevésbé ismert talán, hogy az Android története nem a Google-nál kezdődött. A „hivatalos” történetírásban általában a következőket találjuk:

- 2003-ban alapította az Android Inc.-et 4 ember, Rich Miner, Andy Rubin, Chris White és Nick Sears.
- 2005-ben vásárolta fel a céget a Google.
- 2006-ban jelentek meg az első pletykák, hogy a Google tervezi a belépést a mobilpiacra.
- 2007 végén megalakul az Open Handset Alliance. Az alapító tagok között van: Google, HTC, Motorola, Intel, Qualcomm, Sprint Nextel, T-Mobile és az NVIDIA
- Szintén 2007 végén kiadják az Android SDK első béta verzióját, és bejelentik a 10 millió dollár összdíjazású Android Developer Challenge-et.
- 2008 szeptemberében kiadják az Android 1.0-át és az első telefont a G1-et.

Valójában az Android technológiai gyökerei egészen a BeOS-ig nyúlnak vissza. A Binder rendszert itt kezdte el fejleszteni Dianne Hackborn és kollégái, ami a következő generációs BeOS alapja lett volna. A fejlesztést a PalmSource-nál fejezték be (miután a Palm felvásárolta a Be Inc.-t), és a „PalmOS Cobalt” kódnevű rendszernek alapja volt. Ez az OS végül nem került széles felhasználásra, és a Palm nyílt forrásúvá tette a Binder egy verzióját OpenBinder néven. 2006-ot írtunk ekkor, amikor Dianne Hackborn a Google-nél folytatta a munkát, és ma már tudjuk, hogy a Binder az Android alapvető keretrendszere lett. Kicsit később visszatérünk még a Binderre, hogy részletesebben is megismerjük.

Nagyon izgalmas időszak volt 2007 vége és 2008 eleje. Mindenki próbálgatta az akkor még igen kezdetleges SDK-t, és aki tehette, megpróbálkozott az Android Challenge-en történő részvétellel is.

Szinte naponta jelentek meg új hardverportok, amikor pár lelkes fejlesztő az SDK-t addig reszelte, amíg egy kompatibilis CPU-val rendelkező eszközön sikerült elindítani, és néhány funkcióját kipróbálni. Jómagam egy Sharp Zaurusra portoltam ilyen módon ezt az első nyilvános Android verziót.

Ezt egyébként a közösség tagjai a mai napig csinálják, most az Android 4.0 (Ice Cream Sandwich) SDK portolása a sláger a legkülönfélébb eszközökre. A Google ígérete szerint a Galaxy Nexus megjelenése után nem sokkal elérhető lesz az új platformverzió teljes forráskódja, és megindulhat a valódi portolási munka például a CyanogenMod projekt keretén belül.

2.2. Megjelenik az Android 1.0 és a T-Mobile G1

2008 szeptemberében kiadják az Android 1.0-át és a T-Mobile G1-et. Ezzel elindul az Android platform, hogy meghódítsa a világot. Természetesen elkerülhetetlen, hogy az iPhone megjelenéséhez hasonlítsuk ezt az időszakot. Az iPhone-hoz képest a végfelhasználók részéről sokkal kisebb volt az érdeklődés, ennek egyik oka volt az is, hogy kizárólag a T-Mobile USA hálózatán volt elérhető az első időszakban a készülék, melynek kevésbé jó a lefedettsége, mint az iPhone-t akkoriban kizárólagosan forgalmazó AT&T-nek.

Fejlesztőként viszont mindenképpen új fejezet kezdődött: ez volt az első eset, hogy egy teljes, készülékekre telepíthető, általános felhasználásra alkalmas mobiltelefon platform nyílt forráskódú lett. Jómagam óvatos szkepticizmussal vártam a kódnyitást, azt feltételeztem, hogy a korábbi hasonló esetekhez hasonlóan csak részeket nyit meg a Google, és több hónapos munkába kerül majd egyáltalán emulátorban működésre bírni. Ehhez képest nagyon kellemes meglepetés volt, hogy egy jól dokumentált, 3 paranccsal lefordítható, az emulátorban azonnal működő (1-2 hónappal később már G1-esen is futtatható) rendszert kaptunk.

2.3. 2009: 4 kiadás egy év alatt 1.1, 1.5, 1.6, 2.0

2009-ben egy év alatt 4 új Android verzió jelent meg. A platform elérte a 2.0-ás mérföldkövet, közben már tucatnyi különböző típusú eszközön található az Android hivatalosan (és még többön nem hivatalosan).

2.4. 2010: Az okostelefon piac meghódítása

2010-re a korábban szkeptikus hangoknak is el kellett ismerniük, hogy az Android vezető tényezővé vált, és az okostelefonok piacának jelentős szegmensét tekintheti magáénak. Ebben segíti az, hogy a platform gyártófüggetlen, így a felhasználók sok különböző gyártó rengeteg féle készülékéből válogathatnak.

2.5. 2011: Tabletek és Jégkrémszendvics

Idén jelentek meg az első Android-alapú tabletek, amelyek már az Android 3.x-et (Honeycomb) futtatják. A nagyobb képernyőterülethez új felületet terveztek a Google mérnökei. Azonban, hogy tartani tudják a lépést az iPaddel, gyorsabban ki kellett adniuk az új verziót, minthogy olyan rendszert készíthettek volna, ahol a telefonokhoz szánt felület és a tabletekhez szánt felület egymás mellett működhetett volna. Ezért úgy döntöttek, hogy csak az Ice Cream Sandwich forráskódját adják ki, mely már összeolvasztja a két fejlesztői ágat (2.x és 3.x), és a képernyőmérettől függően az annak megfelelő felülettel lehet használni. Ez minden jel szerint néhány héten belül várható, amint a Galaxy Nexus a boltokba kerül.

Közben az Android részesedése tovább nő az okostelefon piacon is. A Nielsen 2011 3. negyedévre vonatkozó adatai alapján az USA okostelefon piac 43%-át Android eszközök uralják, míg az iOS eszközök 28%-on, a BlackBerry 18%-on, a Windows Phone 7%-on állt.

3. Az Android 4.0 (Ice Cream Sandwich) újdonságai

A több szempontból is nagyon várt Ice Cream Sandwich platform sok újdonságot hoz. Tekintsük át röviden, a teljesség igénye nélkül, hogy mit várhatnak a felhasználók és a fejlesztők.

3.1. Felhasználók

- A nemrég használt alkalmazások vizuális megjelenítése
- A Home Screen-re helyezhető mappák és kedvenc alkalmazások dokkoló
- Átméretezhető, interaktívabb app widgetek – akár e-mailt is olvashatunk külön alkalmazás indítása nélkül
- Több funkció érhető el a telefon lezárt állapotában is: jelzések, kamera, zenelejátszó
- Gyors SMS válasz bejövő hívásokra közvetlenül a zárolási képernyőről
- Továbbfejlesztett szövegbevitel és helyesírás-ellenőrzés
- Hanggal vezérelt szövegbevitel, folyamatos diktálás
- A hálózati forgalom finomhangolása forgalomdíjas előfizetéssel rendelkezők részére
- Továbbfejlesztett kamera, galéria alkalmazás szerkesztőfunkciókkal
- Valósidejű videoeffektek
- Egységesített naptár
- Android Beam – NFC alapú adatmegosztás
- Wi-Fi Direct – közvetlen Wi-Fi kapcsolat közelben levő eszközökkel
- Bluetooth Health Device Profile – Az Android kapcsolódhat egészségügyi berendezésekhez, szenzorokhoz adatgyűjtés céljából

3.2. Alkalmazásfejlesztők

- Az Android 3.x korábban csak tableteken elérhető funkciói elérhetők mobiltelefonokon is, többek között:
  - Hardveresen gyorsított 2D grafika
  - Többszörös kijelölés, drag & drop, vágólap
  - Renderscript 3D grafika amely potenciálisan lehetővé teszi komplex megjelenítési feladatok a GPU-n történő végrehajtását
  - HTTP Live streaming, Bluetooth A2DP és HSP profilok, RTP protokoll támogatása
  - DRM keretrendszer
  - Továbbfejlesztett bevitel: billentyűzet, egér, gamepad, joystick támogatás
  - Üzleti funkciók: Teljes titkosítás és házirendek a titkosított tárolás és jelszavak kezelésére

- Szolgáltatófüggetlen Social API, amellyel egyszerre több hálót kezelhetünk egységes módon
- Media streaming API – Eddig nem volt arra lehetőség, hogy egy alkalmazás saját maga kezelje a lejátszani kívánt média adatátvitelét, pl. egy saját protokoll használatával. Mostantól lehetővé válik, hogy MPEG2TS formátumban közvetlenül streamet küldjünk a lejátszórendszernek.
- Text Service API – Lehetővé vált külső helyesírás-ellenőrzők integrálása a platformba, így akár a Hunspellt is használhatjuk majd.

3.3. Platformfejlesztők

A legnagyobb újdonság értelemszerűen az, hogy végre ismét hozzáférhetünk majd a teljes Android platform forráskódjához. Emellett a Google fejlesztők már azt is megosztották velünk, hogy az elmúlt 1 év alatt jelentősen meghízott a platform, így a fejlesztői gépekkel szemben támasztott elvárások is megnövekedtek:

- 6 GB letöltés
- 25 GB tárhely egy buildhez
- 80 GB tárhely az összes AOSP konfiguráció fordításához
- 16 GB RAM ajánlott, ha kevesebb van, akkor javasolt SSD használata
- 5+ óra CPU-idő egyetlen buildhez

A fenti összefoglaló korántsem teljes, ennél sokkal több mindent találhatunk az Android fejlesztői oldalán (http://developer.android.com).

4. Szemelvények az Android platform belsejéből

4.1. Az Android Open Source Project – első lépések

Az AOSP projektet a Google hozta létre, amelynek fő feladata a platform nyílt forrású kiadásainak kezelése, és lehetőség biztosítása arra, hogy külső fejlesztők hozzájárulhassanak a platformhoz. A projekt weboldalán (http://source.android.com) elérhető részletes dokumentáció, amellyel bárki nekivághat a platformfejlesztésnek.

A forráskód letöltése és lefordítása néhány parancs segítségével könnyen megtehető egy modern Linux (Ubuntu LTS a támogatott) vagy Mac OSX 10.6 rendszeren.

Az AOSP a git verziókezelőt használja, de a projekt méretéből fakadóan nem egy darab repositoryt használ, hanem komponensenként / alrendszerenként egyet, összesen kb. 160-170-et a Gingerbread (2.3.x) kiadásban. Ezeknek az együttes kezelését a repo segédprogram végzi a következő módon.

1. Letöltjük a repo shell szkriptet és elhelyezzük egy, a PATH változóban található könyvtárba:

```
curl https://dl-ssl.google.com/dl/googlesource/git-repo/repo > ~/bin/repo
```

2. Létrehozzuk a könyvtárat, ahova le kívánjuk tölteni a forráskódot:

```
mkdir droid ; cd droid
```

3. Inicializáljuk a repo eszközt:

```
repo init -u https://android.googlesource.com/platform/manifest
```

   Az itt megadott URL-en egy git repository található, ami a repo manifest fájlokat tartalmazza. Alapértelmezésben a master branchen található default.xml elnevezésű fájlt használja, ez természetesen parancssori paraméterekkel felülírható. Ebben a fájlban vannak felsorolva a git repositoryk, valamint az elérési út ahová a forráskódba be kell illeszteni az egyes repositorykat.

4. A következő parancs hatására megkezdődik a forráskód letöltése (illetve meglevő letöltés esetén frissítése):

```
repo sync
```

   Ez a művelet az internetkapcsolat sebességétől függően hosszú ideig is eltarthat, mivel több GB adat letöltésére van szükség.

A folyamat végén a gépünkön elérhető lesz az Android teljes forráskódja a nyilvános kódtörténettel együtt. Ezután már csak le kell fordítanunk a rendszert. Az AOSP weboldalán megtalálható azon csomagok listája, melyeket telepítenünk kell, hogy megkezdhessük a fordítást.

A telepítés után csupán 3 parancsot kell kiadnunk:

```
. build/envsetup.sh
lunch
make -j 4 2>&1 | tee b-1.log
```

A lunch parancs segítségével konfigurálhatjuk, hogy milyen eszköz számára szeretnénk lefordítani a rendszert. A mindenki számára működő lehetőség természetesen az emulátor, de a Gingerbread kiadásban lehetőségünk van a Nexus One (passion) és a Nexus S (crespo) kiválasztására is. Ha nem emulátor számára fordítunk, akkor szükségünk lesz még a fordítás megkezdése előtt néhány olyan fájlra, melyek csak a készülékről, vagy egy hivatalos frissítőcsomagból érhetők el. Ezek általában firmware fájlok, illetve zárt forrású driverek (pl. OpenGL, GPS vagy a GSM modem). A fájlok kigyűjtéséhez a projektben elérhető szkripteket használhatjuk, részletek az AOSP honlapján olvashatók.

A fordítás eredményét az out/target/product/productname/ könyvtárban találjuk.

Innen (amennyiben emulátorra fordítottuk a rendszert) már csak az emulator parancsot kell kiadnunk, és ki is próbálhatjuk a frissen fordított Android rendszerünket. Ha hardveren szeretnénk futtatni, akkor a feltöltés leggyorsabb módja a fastboot parancs használata:

```
fastboot flash boot boot.img
fastboot flash system system.img
fastboot flash recovery recovery.img
fastboot erase cache
fastboot erase userdata
```

Amennyiben már van az eszközünkre egy kényelmes recovery rendszer telepítve (pl. a ClockworkMod recovery), nem szükséges azt felülírnunk.

Ejtsünk néhány szót az elkészült image-ekről / partíciókról:

- *boot*: Ez az image tartalmazza a kernelt és az initrd-be csomagolt root fájlrendszert
- *system*: Ez az image tartalmazza a /system tartalmát, gyakorlatilag az összes rendszerprogramot és előretelepített alkalmazást.
- *userdata*: Ezen a partíción találhatók a felhasználói adatok, melyek a futó rendszeren a /data alatt találhatók. Ide kerülnek az utólag telepített alkalmazások is, valamint minden alkalmazásnak itt található a saját privát könyvtára is.
- *cache*: Átmeneti adatok tárolására szolgáló partíció, például ide tölti le a rendszer a szoftverfrissítéseket vagy a weboldalakról letöltött fájlokat.

A userdata és a cache partíciók tartalma nem minden esetben kompatibilis a különböző Android rendszerverziókkal. (A hivatalos frissítéseknél persze figyelnek arra, hogy ilyen gond lehetőleg ne legyen.) Ha ilyet tapasztalunk, pl. alkalmazások nem indulnak, akkor ezen partíciók tartalmának a törlése jelenthet megoldást. Ezt szokták „data wipe”-nak is hívni.

Amennyiben készen állunk arra, hogy megtegyük első módosításunkat, létre kell hoznunk egy feature branchet, ahova dolgozni fogunk:

```
repo start branchneve [repo1 repo2 ....]
```

Alapértelmezésben ez minden git repositoryban létrehozza a branchet, de korlátozhatjuk csak megadott alrendszerekre.

A módosításainkat a szokott módon tehetjük meg és commitolhatjuk a git segítségével. Amikor elkészültünk a módosításunkkal, a repo upload parancs segítségével tölthetjük fel az Android code review rendszerébe.

Az Android projekt a Gerrit nevű rendszert használja, amit eredetileg a repo-val együtt az Androidhoz fejlesztettek ki, de ma már egyéb projektek is használják. Integrálja a git repositoryk kezelését, a beküldött módosítások ellenőrzését és közvetlen commitolását. Emellett sokféle rendszerrel képes integrálódni, például a felhasználók kezelésére: OpenID, LDAP ... stb. Szintén nyílt forráskódú projekt (a repo-val együtt). Az Android eszközöket készítő cégek jelentős része ezt használja a saját belső fejlesztéseik koordinálására.

Az AOSP-ben a kommunikáció levelezőlistákon (Google csoportok) zajlik, itt kérhetünk segítséget, ha elakadnánk valahol:

- android-platform: Általános fejlesztői kérdések a platformmal kapcsolatban
- android-contrib: Konkrét beküldendő vagy beküldött módosításokkal kapcsolatos lista
- android-building: A platform fordítása során keletkező problémák megbeszélése
- android-kernel: Az Android Linux kernellel kapcsolatos beszélgetés
- android-porting: Az Android új eszközökre portolásával kapcsolatos kérdések
- repo-discuss: A repo és Gerrit eszközök fejlesztése

Minden listát olvasnak a Google fejlesztők és aktívan részt vesznek a beszélgetésben / támogatásban is.

Magyar nyelvű segítséget az android-hu Google csoportban kaphatunk.

4.2. Folyamatok, szolgáltatások és ami mindent összeköt: Binder

A Binder felelős azért, hogy a különböző Android folyamatok / szolgáltatások kommunikálni tudjanak egymással. Hasonló technológiák nem ismeretlenek a hagyományos Linux rendszerekben sem. A modern Linux desktopok hasonló célokra a D-BUS-t használják, de sokkal kisebb mértékben, mint az Android a Bindert. A Binder főbb tulajdonságai:

- *Objektum-orientált megvalósítás*: A Binderben interfészeket megvalósító objektumok kommunikálnak egymással távoli metódushívások segítségével, nem csak strukturálatlan üzenetküldés történik.
- *Objektum azonosítás*: Lehetőség van arra, hogy egy Binder objektumreferenciát továbbküldjünk egy másik objektumnak. Ezt akárhányszor tesszük is meg, a referencia mindig az eredeti Binder objektumra fog mutatni. Így lehetőségünk van az objektumreferenciákat azonosságellenőrzésre is használni.
- *Biztonság*: A Binder hívások feldolgozásakor ellenőrizhetjük, hogy a hívó félnek milyen jogosultságai és a folyamatának milyen uid-je van. Ez a két eszköz képzi az Android biztonsági keretrendszerének alapját.
- *Teljesítmény*: A Binder elég gyors ahhoz hogy az egész Android rendszert rá építsék. Ráadásul lehetőség van file descriptorok átadására is, melynek segítségével memóriaterületek, nyitott fájlok vagy socketek oszthatók meg könnyen és biztonságosan a folyamatok között. Így van arra lehetőség, hogy a SurfaceFlinger (a grafikus hardvert kezelő komponens) lefoglaljon egy memóriaterületet egy ablak számára, amit átad a Window Manager-nek. A Window Manager ezt a területet továbbadhatja az ablakot igénylő alkalmazásnak, így az alkalmazás közvetlenül a videomemóriába tud rajzolni anélkül, hogy közvetlen jogosultságot kéne adni számára a videomemória eléréséhez.

Minden objektumorientált IPC (inter-process communication) megoldásnál 3 fogalommal kell megismerkednünk:

- *Interface*: Az interfész meghatározza azokat a műveleteket, amelyeket a távolról meghívható objektumunkon végezni tudunk.
- *Proxy*: A hívó oldalon található, megvalósítja az interfészt, és a metódushívásokat az IPC rendszeren keresztül átküldi a távoli objektumhoz.
- *Stub*: Fogadja az IPC rendszertől érkező kéréseket, átalakítja a helyi metódushívásokká, valamint az IPC rendszeren keresztül visszaküldi a hívó fél számára az eredményt.

A Bindert lehet használni az Android SDK használatával Javaból, vagy közvetlenül a platform kódból C++-ból. Az SDK része az aidl segédeszköz, amely egy AIDL nyelvű (Javara nagyon hasonlító) interfész leírásból legenerálja a stub és proxy osztályokat Java nyelven. Sajnos C++-hoz kézzel kell megírnunk ezeket az osztályokat, de mint látni fogjuk, ez sem nehéz.

A következőkben bemutatok egy egyszerű összeadó szolgáltatást, ahol a paraméterekben átadott számok összegét kapjuk vissza eredményben.

Az interfészünk így néz ki:

```
class IAddService: public IInterface
{
public:
  DECLARE_META_INTERFACE(AddService);
  virtual int32_t          add(int32_t a, int32_t b) = 0;
```

```
};

enum {
  ADD = IBinder::FIRST_CALL_TRANSACTION
};
```

A Proxy osztályunk:

```
class BpAddService: public BpInterface<IAddService>
{
public:
  BpAddService(const sp<IBinder>& impl) : BpInterface<IAddService>(impl)
  {
  }

  int32_t add(int32_t a, int32_t b)
  {
    Parcel data, reply;
    data.writeInterfaceToken(IAddService::getInterfaceDescriptor());
    data.writeInt32(a);
    data.writeInt32(b);
    remote()->transact(ADD, data, &reply);
    return reply.readInt32();
  }
};
```

A Stub osztályunk:

```
class BnAddService: public BnInterface<IAddService>
{
public:
  virtual status_t    onTransact( uint32_t code,
                                  const Parcel& data,
                                  Parcel* reply,
                                  uint32_t flags = 0) {
    switch (code)
    {
    case ADD:
      {
        CHECK_INTERFACE(IAddService, data, reply);
        int32_t a = data.readInt32();
        int32_t b = data.readInt32();
        int32_t result = add(a, b);
        reply->writeInt32(result);
        return NO_ERROR;
        break;
      }
    default:
      return BBinder::onTransact(code, data, reply, flags);
    }
  }
};
```

Végül természetesen implementálnunk kell az interfészt megvalósító osztályt is:

```
class AddService: public BnAddService
{
public:
  virtual int32_t    add(int32_t a, int32_t b) {
      return a + b;
  }
};
```

Mint látható a fentiekből, egy távoli metódushívás tranzakció összeállításához csupán az adott interfész azonosítóját, a metódus belső (csak a proxy és stub implementáción belül értelmezett) kódját, és a metódushívás paramétereit kell összeállítanunk, illetve a szerver oldalon feldolgoznunk.

Lássunk egy konkrét példát a Binder használatára. Egy modern operációs rendszernek fontos, hogy jó multimédiás képességei legyenek, ideértve a hang- és videofelvétel és -lejátszás kezelését. Android rendszereken ezt a mediaserver folyamat végzi. A mediaserveren belül több szolgáltatást is találunk:

- *AudioFlinger*: a különböző hangforrások mixelését és a hardverhez juttatását végzi, illetve kezeli a hangfelvételt is.
- *Stagefright*: általános médialejátszó és felvevő keretrendszer

Adódik a kérdés, hogy a Java nyelven írt alkalmazásokban használható MediaRecorder és MediaPlayer osztályok hogyan kapcsolódnak a mediaserverhez. A megoldást ismét a Binder framework jelenti. Mind a MediaPlayer mind a MediaRecorder Java osztály egy-egy Binder interfészhez kapcsolódik, melyeket a mediaserver szolgáltat. Kihasználva a Binder azon tulajdonságát, hogy file descriptorokat adhatunk át közvetlenül más folyamatoknak, lehetővé válik, hogy a mediaserver a Java alkalmazás által megnyitott fájlokat játsszon le.

4.3. Az Android portolása új hardverre

Azt a technológia iránt érdeklődő emberek többsége tudja, hogy az Android a Linux kernelre épül. Tehát minden rendszeren, ahol van működő Linux változat, fut az Android is – gondolhatnánk. A helyzet azonban nem ennyire egyszerű, mivel az Android a hagyományos Linux (GNU / Linux) rendszerektől különböző userlandet (a kernel által futtatott felhasználói programok összessége) használ.

Ide tartozik a fentebb részletesen bemutatott Binder, de a libc is egy BSD gyökerekkel rendelkező, a GNU/Linux rendszereken általában használt glibc-vel binárisan nem kompatibilis könyvtár. Ez azt is jelenti, hogy minden programot, melyet egy Android rendszeren szeretnénk futtatni, újra kell fordítani.

Lássuk hát részletesen, mi szükséges ahhoz, hogy egy új hardveren használatba vehessük az Androidot:

- Kell egy Android patcheket tartalmazó, megfelelő verziójú Linux kernel. Ezek a patchek tartalmazzák többek között a binder kernelbeli részét, az ashmem osztott memória drivert, és a különféle, energiagazdálkodással kapcsolatos módosításokat. Beágyazott rendszereknél a „megfelelő” kernelverzió beszerzése sem mindig egyszerű, mivel nem minden gyártó küldi be a hardvereik támogatását a hivatalos „vanilla” kernelfába. Így sokszor arra vagyunk utalva, hogy a chipset-gyártó mikor készít el egy Androidhoz is megfelelő verziójú kernelt.
- A legfontosabb (assembly kódokat is tartalmazó) komponensek portolása a hardver processzorához. A teljesség igénye nélkül:: libc, Dalvik (az Android alkalmazások virtuális gépe), Skia (2D rajzolási feladatok). Szerencsére a 3 legelterjedtebb processzorcsaládhoz (ARM, MIPS, x86) már van működő változat, így ezekkel a legtöbb esetben nem kell foglalkoznunk.

- A portolás talán leglátványosabb része, amikor a megjelenítést végző komponenst (SurfaceFlinger) portoljuk. Sajnos ez okozza legtöbb problémát is, mivel sok esetben a grafikus chip dokumentációja nem hozzáférhető, és a driverkönyvtárak csak bináris formában érhetők el. Ilyen esetben a chip beszállítóján múlik, hogy kiad-e Android drivert, illetve milyen gyorsan frissíti a drivert egy új Android verzió megjelenése után.

- Hasonlóan problémát jelent a videolejátszásnál használt codecek portolása. Itt is csak a legritkább esetben érhető el a gyorsítóchipek dokumentációja, tehát ismét a hardvergyártóra vagyunk utalva.

- Ezután következik az egyéb hardver driverek portolása: hang, GPS ... stb. Itt a legtöbb esetben nem a kernel driverekre kell gondolni, hanem a felhasználói programként futó részekre, pl. ALSA driver illesztés az Android frameworkhöz, vagy egy GPS daemon.

4.4. Hová tűnt a forráskód: a kernel.org incidens

Mint az ismeretes augusztus közepén kiderült, hogy feltörték a kernel.org infrastruktúra egyik központi szerverét. Emiatt a korábban innen kiszolgált Android forráskód is elérhetetlenné vált. A Google alkalmazottak „nincs még semmi bejelenteni valónk" válaszai nem segítettek eloszlatni az aggodalmakat, hogy mikor válik a forráskód újra elérhetővé.

Végül az Android 4.0 bejelentésekor megkaptuk a választ: az AOSP forráskódokat mostantól a Google a saját szerverein keresztül teszi elérhetővé, nem veszik igénybe a kernel.org segítségét. Állításuk szerint már a biztonsági incidens előtt dolgoztak az átálláson, hiszen a kernel.org infrastruktúra nehezen birkózott már meg az új Android verziók megjelenésekor jelentkező igényekkel, amelyek a kernel.org többi szolgáltatására is negatív hatást gyakoroltak.

A feltörés után ezért úgy döntöttek, hogy meg sem próbálják a régi rendszeren újraindítani a szolgáltatást, hanem megvárják, amíg elkészül a saját kiszolgálórendszer.

5. A sötét oldal

Ebben a fejezetben áttekintjük az Android árnyoldalait, problémáit, melyeknek megoldása még várat magára.

5.1. Elég nyílt-e az Android?

Ez a téma talán az egyik legkedveltebb mostanság (közvetlenül a biciklitároló (bikeshed) színének megvitatása után). Amikor valaki azt mondja, hogy az Android nem eléggé nyílt, általában 2 dolgot ért alatta.

- Nincs hivatalos útiterv-/ ütemterv, tehát nem lehet előre tudni, hogy mikor érkezik új verzió, és abban milyen új funkciók mutatkoznak be.

- Ahhoz, hogy valódi hardverekre tudjuk telepíteni az Androidot, szükség van zárt forrású komponensekre.

Vegyük át kicsit részletesebben, hogy ez mit jelent.

Útiterv nélkül

A Google-nek általában az az üzletpolitikája, hogy nem nyilatkozik ütemtervekről és funkciótervekről. Természetesen ez sincs kőbe vésve, például a Google App Engine-nél a többi Google termékhez képest sokkal részletesebb ütemtervet találunk. Ez a stratégia a Google szempontjából érthető:

- Csúszás, vagy tervezett funkciók kiesése esetén nem kell kellemetlen sajtótájékoztatókat tartani.
- Jól irányzott kiszivárogtatásokkal / pletykákkal hatékonyan fent lehet tartani az érdeklődést hivatalos nyilatkozatok nélkül is.

Ettől természetesen még nem lesz kevésbé frusztráló ez az üzletpolitika mindenki más számára.

Az Android esetében azt láttuk, hogy negyedéves pontossággal mindig behatárolták a következő kiadás megjelenésének időpontját, valamint körülbelül 1 hónappal az új verziót futtató első készülékek megjelenése előtt már elérhető volt az új SDK verzió, így a fejlesztőknek elég idejük lehetett arra, hogy az alkalmazásaikat frissítsék.

Az AOSP forráskód kiadásánál már nem ment minden ennyire zökkenőmentesen. Az 1.x-es változatoknál még viszonylag gyorsan elérhető volt a készülék megjelenése után, de a 2.x időszakban előfordult több hónapos várakozás is, és a 3.x-es platformváltozatok forráskódja valószínűleg soha nem lesz elérhető a nyilvánosság számára.

Itt érdemes egy kis kitérőt tennünk a különböző nyílt forrású fejlesztési modellek felé. Amikor nyílt forrású projektről beszélünk, akkor nagyon gyakran arra gondolunk, hogy nem csak a projekt eredménye (a stabil kiadások) nyílt forráskódúak, hanem a teljes fejlesztési folyamat is nyíltan folyik. Talán a két legismertebb ilyen jellegű projekt a Linux kernel fejlesztése és a Debian Linux fejlesztése.

Ezzel szemben egy másik modellt képvisel a kereskedelmi nyílt forrású projekt, ahol egy profitorientált cég adja ki a fejlesztés eredményeit nyílt forrású termékként. Ilyenekre sok példát találunk pl. az üzleti szoftverek piacán az OpenERP és az OpenBravo, vagy az ExtJS javascript keretrendszer. Ebbe a csoportba sorolható a RedHat Enterprise Linux disztribúciója is.

A kereskedelmi nyílt forrás (commercial open-source) modellben gyakori, hogy csupán a végleges verziók érhetők el nyílt forrású termékként, míg a fejlesztői változatokat csak egy szűkebb kör (általában a kiemelt ügyfelek) ismerhetik meg. A fejlesztés túlnyomó részét ilyen projektek esetében a forráskód felett jogilag ellenőrzést gyakorló szervezet fejlesztői végzik, és a külső hozzájárulók inkább kisebb javításokat, illetve kiterjesztéseket készítenek. A fejlesztés irányvonalát is a jogtulajdonos határozza meg, üzleti érdekei mentén.

Az Android projekt egyértelműen az utóbbi csoportba tartozik. Ha így tekintjük át a projekt folyamatait, akkor egy átlagos „commercial open-source” projektet látunk, az átlagnál jobb eszközökkel, melyek segítik a külső hozzájárulók munkáját.

Természetesen, mint Android fejlesztő és felhasználó is örülnék annak, ha több információ lenne publikus a készülő új funkciókról. Azonban reálisan gondolkodva egy ilyen lépés sértené mind a Google, mind készülékgyártó partnerei érdekeit. A mobilpiacon nagyon éles a verseny, és néhány hónap leforgása alatt teljesen átrendeződhet a helyzet. Erre maga az Android a legjobb példa, hiszen 2 év alatt vált a semmiből piacvezetővé. Ugyanez fordítva is lejátszódhat bármikor.

Zárt forrású komponensek

Kétségtelen tény, hogy a modern hardvereszközök dokumentációjához csak a legritkább esetben férhetünk hozzá titoktartási nyilatkozat nélkül, amely az esetek túlnyomó többségében kizárja a nyílt forrású driver megvalósítást. Ez a beágyazott rendszerekre hatványozottan igaz. Sok esetben még a partnerek is csupán az egyes driverek, pl. az OpenGL driver bináris verzióját kaphatják meg.

A jogszabályi környezet és a piaci folyamatok miatt valószínűtlen ennek a helyzetnek a megváltozása belátható idén belül.

5.2. Elárvult készülékek, a platform fragmentációja

Bejárta az Internetet az understatement.com cikke, amely jól bemutatja, hogy milyen problémák vannak az Android készülékek operációs rendszer támogatásával. Sajnos, ellentétben az iPhone-nal, az Android esetében nem biztosított az, hogy egy megvásárolt készülék az átlagosan 2 éves hűségnyilatkozat lejártáig mindig a legfrissebb szoftvert futtassa.

Tény, hogy itt az Apple stratégiája rengeteg előnyt jelent. Az iPhone-nál mindent az Apple irányít (beleértve a hardvert és a szoftvert is), így könnyebben tudja a régebbi eszközöket is támogatni. A másik jelentős különbség, hogy az iPhone esetében a készülékgyártó elemi érdeke, hogy a készülékeket használni tudják tulajdonosaik, hiszen ekkor fognak az alkalmazásboltban bevételt generálni.

Az Android esetében (tudomásom szerint) a készülékgyártók nem kapnak közvetlen részesedést az alkalmazások értékesítéséből befolyt bevételből, így nem érdekeltek a már eladott készülékek további támogatásában. Amíg ez a helyzet nem változik, sajnos nem számíthatunk számottevő javulásra.

Természetesen a Google is látja a problémákat, és megpróbálnak különböző eszközökkel hatni a készülékgyártókra, például, akik jobb támogatást vállalnak, azok hamarabb férhetnek hozzá a fejlesztés alatt álló és még nem publikus Android forráskódokhoz.

Érdemes azt is figyelembe venni, hogy az Android piac összetétele teljesen más, mint az iPhone esetében. Minden egyes iPhone változat kiadásának pillanatában hardver szempontból felső kategóriás készüléknek számított, amelyek még több év elteltével is elfogadható teljesítményt tudnak nyújtani.

Ezzel szemben az eladott Android készülékek jelentős része „belépő szintű", kisebb erejű, kifutó hardverelemekre épül, ez teszi lehetővé, hogy valóban fillérekért megvásárolhatók legyenek. Ez azt is jelenti, hogy egy ilyen, technológiai szempontból már megjelenése pillanatában elavult eszköznél a több évig folyamatos rendszerfrissítés biztosítása csak jelentős többletköltséggel és fájó kompromisszumok árán lenne megoldható.

Ennek a helyzetnek a megoldása közel sem triviális. Például a Google megtehetné, hogy jobban megköti a gyártók kezét, akik a Google alkalmazásokat (beleértve az Android Marketet) is terjeszteni szeretnék, és megkövetelik azt, hogy csak közép- és felsőkategóriás hardverrel rendelkező eszközöket dobnak piacra. Ez azonban csak tovább rontaná a helyzetet:

- Az Android jelentős piacvesztést szenvedne el a belépő szintű okostelefonok piacán, akár (a már folyamatosan temetett) Symbian, akár egyéb proprietary platformok, mint pl. a Samsung Bada előnyére.
- Az Android nyílt forráskódú volta miatt gyártók továbbra is készíthetnének belépő szintű eszközöket, ezek azonban a Google alkalmazások nélkül kerülnének a felhasználókhoz, amely korántsem nyújtaná a megszokott élményt, és sok bosszúságot szülne a vásárlók számára. Ez megint csak negatívan befolyásolná a platform megítélését

Egy másik lehetséges megoldás lehetne a belépő szintű, valamint a közép- és felsőkategóriás készülékek jobb megkülönböztetése, pl. Android Light vagy Android Gold jelzések használata.

Az itt vázolt okok a platform fragmentációjához vezetnek, ami lassítja az új funkciók terjedését, hiszen a fejlesztők kénytelenek korábbi platformverziót célozni, hogy maximalizálni tudják a kompatibilis eszközök számát. Az egyetlen kiútnak az tűnik, ha a Google-nek sikerül az eszközgyártókat érdekeltté tenni a korábban eladott készülékeik támogatásában.

6. Kitekintés a jövőbe

Az biztosnak látszik, hogy az Android helyzete a közeljövőben tovább fog erősödni, különösen az új platformverziónak köszönhetően, amely lehetővé teszi, hogy mobiltelefonok mellett hamarosan

tv-készülékeket és set-top boxokat is célozzunk alkalmazásainkkal.

Hogy innen merre vezet az út, még elég homályos. Egyelőre úgy tűnik, hogy az Android nem fogja leváltani az általános célú operációs rendszereket, mint a Windows, Linux vagy a Mac OS X. Ugyanakkor az átlagfelhasználók, akik főleg internetes alkalmazásokat használnak, egyre kevésbé igénylik ezeknek a rendszereknek az extra szolgáltatásait.

Szintén a Google-nél fejlesztik a Chrome OS-t, amely kizárólag ilyen webes alkalmazások futtatására képes. Középtávon várható, hogy a két projekt egyesül, hiszen mindkét rendszer ugyanazt a szegmenst célozza, és egyesítve erőiket jobb felhasználói élményt nyújthatnának.

Az Android fontosságát jól mutatja, hogy bekerült a beágyazott chipsetek kötelezően támogatandó operációs rendszerei közé a Linux és a Windows CE mellé.

Egy biztos: az Android szerepe a belátható jövőben továbbra is meghatározó lesz.

7. A szerzőről

A MattaKis Consulting alapítója és ügyvezetője. Az elmúlt években mobiltelefonoktól a J2EE vállalati alkalmazásokig számos projektben volt fejlesztő, architekt és projektvezető. 14 éve Linux felhasználó és fejlesztő. Aktív támogatója a nyílt forráskódnak, korábban a Linux-felhasználók Magyarországi Egyesületében, most a Magyar Android Közösségben. Feleségével és kisfiával Dunakeszin él.

A KDE jövője

Kiszel Kristóf

Tartalomjegyzék

1. Előszó **92**

2. A Trolltechtől a Nokián keresztül a nyíltságig **92**
2.1. A Qt Project . . . 93
2.2. A trendek követése: Qt 5 . . . 93
2.3. A jövő víziója . . . 94

3. A jövő már itt van **94**
3.1. Ez már a KDE5? . . . 94
3.2. A KWin ablakkezelő . . . 95
3.3. Túl az X11-en: út a Wayland felé . . . 95
3.4. QML a Kwinben . . . 96
3.5. Plasma Active: határ a csillagos ég . . . 96
3.6. Ádámtól és Évától: egy kis nevezéktan . . . 96
3.7. Plasma Active One . . . 96

4. Összefoglalás **97**

1. Előszó

A KDE a három nagy asztali környezet egyikeként él sokak szemében, mára azonban jóval többé vált 15 évvel ezelőtti önmagához képest. Nem csak egységes környezetet nyújt az ablakkezeléstől kezdve a felhasználók igényeit lefedő alkalmazásokig, hanem különböző eszközökre optimalizált munkaterületeket és fejlesztői platformot is. Fejlődése azonban mindig is függött a Qt keretrendszertől, amely a KDE alapját adja, és kezdetben még csak nem is volt szabad szoftver, szemben a GNOME-mal. Később azonban ez megváltozott, a Qt jelentős fejlődésen ment át nem csak a képességeit, hanem a fejlesztési modellt is tekintve. Ahogy korábban szó volt róla, a Qt és a KDE mindig is összefonódott, ezért nem lehet úgy beszélni az egyikről, hogy ne ejtenénk szót a másikról is. A továbbiakban tehát áttekintés következik a Qt történetéről, majd a 2012-ben érkező következő főverziójáról, végezetül rátérünk a lényegre, azaz a KDE jövőjére.

Mi is a KDE jövője? A kérdés talán homályos első hangzásra, sokféle elképzelés születhet az Olvasó fejében a cím mögött lévő tartalomról, íme hát egy kis pontosítás: a Qt történeti áttekintése után, abból kiindulva bemutatásra kerül néhány azok közül a fejlesztések közül, amelyek vagy már elkezdtek beérni, vagy még csak folyamatban vannak. A KDE hatalmas rendszer, a több mint másfél millió kódsor egyes területei stagnálnak, mások fokozatosan fejlődnek, megint mások radikális irányváltáson mennek keresztül, és újak is csatlakoznak. A számítástechnika világa folyamatosan változik, új kihívások jelentkeznek, amelyekre a KDE fejlesztői igyekeznek megoldást nyújtani, ilyen kihívás például a tabletek és okostelefonok robbanásszerű elterjedése, melyre adott válasz a Plasma Active. Hogy a személyi és hordozható számítógépek se maradjanak ki, utolsó témaként szó lesz az idén tizenkettedik születésnapját ünneplő Kwin ablakkezelőről.

2. A Trolltechtől a Nokián keresztül a nyíltságig

A Qt fejlesztése 1991-ben kezdődött az akkor még Quasar Technologies nevű cégnél, amelyből később a Trolltech lett. Kezdetben csak Unix X11 és Windows platformokra készült el, utóbbira zárt licenc alatt, ami azt jelentette, hogy a UNIX-ra készített szabad és nyílt forrású alkalmazásokat nem lehetett portolni Windowsra a licenc megvásárlása nélkül. A Qt 2.2 2000-es megjelenése volt az egyik fontos mérföldkő, nem kereskedelmi felhasználásra ugyanis ekkortól volt elérhető GNU GPLv2 licenc alatt, addig csak egy ezzel nem kompatibilis licenc (QPL, Q Public License) szerint volt szabadon használható, amelyet a Free Software Foundation elutasított, ami egyben azt is jelentette, hogy az alapítvány a KDE-t nem tekintette szabad szoftvernek. Ezért egyezség született a KDE és a Trolltech között: ha 12 hónapon keresztül nem jelenik meg új szabad/nyílt forrású verzió a Qt-ből, az egyezség által létrehozott KDE Free Qt Foundation egy BSD-stílusú licenc alatt kiadhatja a Qt forráskódját. 2005-ben a Qt 4.0 megjelenésével a windowsos változat is a GPL licenc hatálya alá került, vagyis a kereskedelmi és a szabad változat végre ugyanazokat a platformokat támogatta.

A Nokia 2008-ban vásárolta fel a Trolltechet, a Qt fejlesztése során pedig arra koncentrált, hogy az általa gyártott eszközök fő fejlesztési platformja legyen. A forráskód innentől kezdve a Gitorious-on keresztül volt elérhető, a fejlesztést azonban még mindig a Nokia végezte házon belül. Idén januárban azonban bejelentették, hogy felsőkategóriás telefonjaikon a Microsoft Windows Phone 7 rendszert használják a korábban kiválasztott, az Intellel közösen fejlesztett MeeGo helyett. A szabad szoftveres közösség azon nyomban felbolydult, hiszen a bejelentés megkérdőjelezte a Nokia és a Qt viszonyát, a Nokia azonban kiállt a Qt mellett, és Symbian-alapú telefonjain továbbra is használja, illetve megjelentették az N9-et, az első és utolsó telefont, amely a MeeGo 1.2 Harmattan rendszert alkalmazza. Még egy további fontos dolgot is bejelentettek: nyílttá teszik a Qt fejlesztését még 2011-ben. Ez lett a Qt Project, amelyről az alábbiakban lesz szó.

2.1. A Qt Project

A Qt Project meritokratikus, konszenzus-alapú közössége mindazoknak, akik érdekeltek a Qt fejlesztésében. Bárki csatlakozhat a közösséghez, részt vehet a fejlesztésben és a döntési folyamatokban. Habár a Nokia hívta életre a Qt Projectet, és a közösség tagjainak nagy részét szintén a Nokia adja, a közreműködők 15-20%-a már most külsős, például más cég alkalmazottja. A közreműködőknek öt szintje van, attól függően, hogyan és mivel járulnak hozzá a projekthez. A legalacsonyabb szint a felhasználóké (Users), ők azok, akik blogjukban írnak a projektről, visszajelzéseket küldenek, vagy csak egyszerűen megköszönik a fejlesztők munkáját. Belőlük lehetnek a közreműködők (Contributors), akik már a tényleges fejlesztést végzik, kezdve a dokumentációírástól a hibajavításon át a közösségi munkáig és az új felhasználók támogatásáig. Egy Contributor azonban nem közvetlenül fejleszt, hanem javításokat küld, amelyeket azután a következő szint, az Approver néz át és fogad el. Ők azok, akik kivételes esetben közvetlenül módosíthatják a kódot, ez azonban nagyon ritka, például fordítási hiba esetén fordulhat elő. Az utolsó előtti szint a karbantartóké (Maintainers), akik a Qt egy-egy részmoduljáért felelősek. Fontos, hogy jól rálássanak a Contributorok és Approverek munkájára, folyamatosan készen kell állniuk egy esetleges béta kiadásra. A karbantartók részt vesznek a döntési folyamatokban, elfogadhatják az irányítási modell változtatásait, kezelik a közösségen belüli vitákat, valamint átnézik azokat a javításokat, amelyeket senki más nem nézett át. A legfelső szintet a Chief Maintainer személye képviseli, ő határozza meg a projekt irányvonalát, és hoz döntést azokban a vitákban, amelyekben sem a közösség nem tudott konszenzusra jutni, sem a karbantartók nem jutottak megoldásra.

A Qt Project honlapja október 19-én indult, itt található a wiki, a levelezőlisták, a hibakövető rendszer, a Gerrit forráskód-áttekintő segédeszköz és sok-sok más, a projekthez kapcsolódó dokumentáció vagy jogi nyilatkozat. Az új, nyílt fejlesztési modell első eredménye a valamikor 2012-ben megjelenő Qt 5 lesz, amely nem hoz olyan radikális váltást, mint 2005-ben a Qt 4 a Qt 3 után.

2.2. A trendek követése: Qt 5

Hat éve jelent meg a legutolsó Qt főverzió, ez idő alatt pedig sokat változott a világ. A mobil eszközök száma óriási mértékben megugrott, az okostelefonok és tabletek új kihívásokat állítanak a fejlesztők elé. Azért, hogy ne maradjon le a versenyben, a Qt-nak is meg kell újítania önmagát, és bár a Qt 5 tervezése a nyílt fejlesztési modell bevezetésével együtt zajlott, már május óta tudni, melyek a fejlesztések fő irányvonalai. Az alapokat a már most is meglévő technológiák adják, mint a Qt Quick, QML Scenegraph és a Project Lighthouse. Melyek a fő célok?

- A grafikus processzorok jobb kihasználása, hogy korlátozott erőforrások mellett is gördülékeny és hardveresen gyorsított grafikus megjelenítést nyújthassanak
- Az alkalmazások és azok felhasználói felülete fejlesztési idejének radikális csökkentése és egyszerűbbé tétele a QML és JavaScript segítségével
- Az alkalmazások és a web lehető leghatékonyabb összekapcsolása például webes tartalmak és szolgáltatások beágyazásával a Qt alkalmazásokba
- A portok elkészítéséhez és karbantartásához szükséges komplexitás és kód csökkentése

A Qt 4.x a mai napig tartalmaz olyan kódokat, amelyek megakadályozzák, hogy a Qt legyen a lehető legalkalmasabb keretrendszer a következő generációs alkalmazásokhoz és felhasználói felületekhez. A Qt 5-ben ezeket hátrahagyják, és célzott módosításokat hajtanak végre az alkalmazásprogramozói felületben (API, Application Programming Interface). Azoknak sem kell aggódni, akik még

emlékeznek a Qt 3→Qt 4 váltásra: ezúttal nem ismétlődik meg ugyanaz, a forrásszintű kompatibilitás az esetek nagy részében megmarad, a fejlesztők azonban hiszik, hogy a bináris kompatibilitás megtörésére szükség van, de törekednek arra, hogy az átmenet minél simább legyen.

Szemben az eddigiekkel, a Qt 5 csak a platformok egy szűk halmazára koncentrál első körben (Linux/Wayland, Linux/X11, Windows, Mac), később azonban a közösségen múlik, hogy lesznek-e további támogatott platformok. Ezzel együtt a most még támogatott platformok egy részének (például a kereskedelmi UNIX rendszereknek, mint a Solaris 10 UltraSparc, AIX 6, stb.) támogatása is megszűnik. A cél az, hogy minden platformon a lehető legjobb funkcionalitást biztosítsák, és ebben sokat segít, ha kevés platformot kell támogatni.

2.3. A jövő víziója

A Qt 5 középpontjába a Qt Quick fog kerülni. Habár a hagyományos C/C++ alkalmazások továbbra is működni fognak, a fejlesztők azt várják, hogy a Qt 5-től kezdve minden felhasználói felületet QML-ben írnak, az alkalmazás logikája pedig JavaScriptben készül, és csak kivételes esetben használnak C/C++ kódot. A kompatibilitás érdekében ugyan megmarad a QWidget-alapú programozási modell és API, hosszú távon azonban a QML fogja leváltani. A KDE már most elkezdte az áttérést QML-re, az egyik lépés a Plasma Active, a másik pedig a plazmoidok átírása QML-re, valamint a QML felhasználása a KWin ablakkezelőben.

3. A jövő már itt van

Tizenöt éves története során a KDE mindig akkor váltott főverziót, amikor a Qt. A legutóbbi ilyen váltás 2008-ban történt, amikor megjelent a KDE 4.0, amelyet óriási viták öveztek, hiszen még a fejlesztők szerint sem volt kész, sokan pedig máig siratják a KDE3-at, azt állítva, a KDE4 még mindig nem érte utol tudásban és minőségben. Hogy ez így van-e, az komoly vita tárgyát képezné, koncentráljunk inkább arra, mit tartogat a KDE számára a Qt 5, és mit tartogat a felhasználóknak a KDE.

3.1. Ez már a KDE5?

2008-ban vált híressé az a mondat, miszerint ez még nem a KDE4. A KDE 4.0 megjelenésekor rájuk zúduló kritikák hatására jelentették ki ezt a fejlesztők, akiket kettős kényszer szorított: ahhoz, hogy szélesebb réteg tesztelje rajtuk és néhány kíváncsi érdeklődőn kívül, stabilnak nyilvánított kiadás kellett, viszont ha kiadnak ilyet, akkor szükségszerűen mindenki azonnal használni akarja, nem törődve azzal, hogy még nincs kész minden. A KDE 4.0 körül gyorsan kialakult a 22-es csapdája: az alkalmazásfejlesztők kivártak, hogy még fejlődjön, és többen használják, a felhasználók pedig arra vártak, hogy kedvenc alkalmazásaik új kiadásai megjelenjenek.

Ennek megismétlődését szeretnék mindenáron elkerülni, a már készülő KDE Frameworks 5.0 ezért nem hoz olyan alapvető változásokat, mint a KDE4: a Qt 5-höz hasonlóan inkább a modularizáció, a függőségek letisztázása és a minőség javítása az elsődleges cél, vagyis ugyanúgy megmarad a forrásszintű kompatibilitás, de a bináris kompatibilitás mindenképpen megtörik. Ha történik is valami változás, ami alkalmazás módosítását igényli, az a lehető legkisebb lesz, és automatikus megoldást készítenek rá , hogy a fejlesztőnek minimális erőfeszítésébe kerüljön azt alkalmazni, az esetek többségében azonban elég lesz újrafordítani az adott alkalmazást, és tesztelni, hogy megfelelően működik-e. A változások a felszín alatt fognak történni: osztályról osztályra átnézik a KDE- t, és ha az adott osztálynak jobb helye lenne a Qt-ban, megpróbálják átmozgatni oda, hogy a Qt használói számára is előnyös legyen az adott osztály által megvalósított funkcionalitás. Amennyiben egy osztályról úgy látják, hogy felesleges, akkor meg fog szűnni, esetleg beolvad egy másik osztályba. Így

kisebb, könnyebben karbantartható kódbázishoz jutnak, amelyen belül tisztábbak és egyszerűbbek a függőségek.

Hogy mi várható pontosan a KDE Frameworks 5.0-ban, azt még csak maguk a fejlesztők tudják, a fenti, meglehetősen kevés és általános információ is tőlük származik. Az útiterv azonban készen áll, és biztosak lehetünk benne, hogy a megfelelő pillanatban minden elénk tárul.

3.2. A KWin ablakkezelő

Lássunk most egy olyan konkrét alkalmazást a KDE-n belül, amely szinte egyidős vele, és hasonlóan nagy változásoknak néz elébe. A Martin Gräßlin vezette fejlesztőcsapat aktívan dolgozik azon, hogy folyamatosan jobb és jobb teljesítményt nyújtson a KWin, amellett, hogy új funkciókat építenek bele, vagy egyes funkcióit esetleg más alapokra helyezik. Két példán keresztül kerül bemutatásra a KWin fejlesztői munkájának összetettsége: az első felszínes áttekintést ad arról, hogyan készülődnek az X11 leváltására, a második pedig azt mutatja be, hogyan alkalmazzák a QML-t a KWinben.

3.3. Túl az X11-en: út a Wayland felé

Az X11 azokban az időkben készült, amikor még nem létezett személyi számítógép, a felhasználók terminálon keresztül kapcsolódtak a szerverhez, és ezen a terminálon jelentek meg a szerver által küldött üzenetek, majd az első grafikus felületek. Röviden összefoglalva: az X11 egy kliens-szerver alapú architektúra, amely alapvetően hálózati működésre lett tervezve, a modern disztribúciókban azonban a kliens és a szerver egy és ugyanaz. A felépítéséből adódóan azonban kikerülhetetlen, az X11 fölötti rétegben helyezkedik el az ablakkezelő és kompozitor, valamint a Plasma Desktop (vagy a Plasma Netbook, Plasma Active), amelyek kénytelenek az X11-en keresztül kommunikálni. A kompozitor (továbbiakban KWin) azonban már régóta átvette az X11 feladatait, így az mára lényegében csak egy proxy a kernel és a KWin között. Nyilvánvaló tehát, hogy jobban járnánk, ha kidobnánk ezt a proxyt, és így sokkal letisztultabb felépítést kapunk szemben a mostanival, amely ráadásul az X11 létrejöttekor még nem is létező igények miatt számos kényszermegoldást is tartalmaz. Kérdés azonban, mivel helyettesíthetnénk az X11-et? A válasz a Wayland, amelyet az X grafikus megjelenítő leváltására hoztak létre 2008-ban, az alapjaitól újratervezett felépítéssel és a meglévő technológiákra (Kernel Mode Settings, Graphics Execution Manager) alapozva. A váltás azonban hosszú folyamat, amivel még maga a Wayland sincs készen, a szabad szoftveres világ azonban már elkezdett készülődni. A Compiz és a KWin is hasonló módot választott: az X11-függő részeket modulokba szervezik, így nem csak a Wayland, hanem más grafikus kiszolgálók támogatása is egyszerűen megoldható lesz, hiszen csak meg kell írni az adott grafikus kiszolgálót támogató modult.

A folyamat csak elméletben egyszerű, a gyakorlatban számos kérdés merül fel, az első rögtön az, sikeres lesz-e a Wayland? Egyelőre senki sem használja, a lehetséges problémák még ismeretlenek, a protokoll nincs teljesen definiálva, nincsenek hozzá ablakkezelők, nincsenek alkalmazások, amelyek használják, és kérdéses az illesztőprogram-ellátottsága is. Az X11 olyan tulajdonságai, mint például a kliens-szerver alapú felépítés, nagyon is jól jönnek hálózati használat esetén. A Wayland azonban sokkal hasznosabb mobil platformokon, ahol nincs szükség az X11-re, ezért a KWin Wayland támogatása is elsődlegesen a Plasma Active-ra koncentrál. Martin Gräßlin becslése szerint körülbelül 10-20 év, mire a Wayland teljesen leválthatja az X11-et; addig egymás mellett fog létezni a kettő, mind a maga területén.

3.4. QML a Kwinben

A QML a Qt Markup Language rövidítése, ebből már sejthető, hogy ez egy leírónyelv, a korábban leírtakból pedig tudható, hogy a Qt alkalmazások felhasználói felületének készítéséhez használják. Az elve nagyon egyszerű: a grafikus elkészíti akár egy tervezőprogrammal (Qt Creator, Qt Designer) az alkalmazás felhasználói felületét, a fejlesztő pedig JavaScriptben megírja hozzá a logikát, de a két szerepkör akár egyesíthető is. A QML nagyon egyszerű és könnyen tanulható nyelv, amelyet remekül szemléltet az is, hogy a KDE 4.8-ban a KWin legalább két területen alkalmazni fogja: az új képernyőzárolóban és az ablakváltóban.

A képernyő zárolását jelenleg a Krunner végzi, amely biztonsági szempontból rossz megoldás, például egy billentyű lenyomásakor az aktiválódó képernyőn kis időre láthatóvá válnak az ablakok , és leolvasható a tartalmuk, mielőtt elsötétülne az egész a feloldó párbeszédablak kivételével. A KDE 4.8-ban ezt a feladatot átveszi a KWin, az új feloldó képernyő pedig QML-ben készül. A feladat átvétele egyben azt is jelenti, hogy a régi X11 képernyővédőket nem támogatja többé a KWin, jó hír azonban, hogy helyettük QML-ben készült képernyővédőket lehet majd használni, amiket aztán a közösség megoszthat egymással a kde-look.org és hasonló oldalakon keresztül, hála könnyű elkészíthetőségüknek.

Az ablakváltó effektusok jelenleg szintén C++ nyelven vannak megírva, ami nagyon nehézzé teszi az elrendezések módosítását, új effektusok hozzáadását. A QML-nek köszönhetően ez is jóval egyszerűbbé válik, akár egy órás munkával is szép elrendezések készíthetők, határt csakis a képzelőerő szabhat. A most meglévő öt effektus mellé továbbiak is bekerülhetnek azoktól felhasználóktól, akik éreznek magukban elég késztetést és tehetséget új ablakváltó elrendezések készítésére és beküldésére, hogy aztán a KDE 4.8 megjelenése után sok-sok ember használhassa azokat.

3.5. Plasma Active: határ a csillagos ég

„Egy mobil eszköznek többnek kell lennie, mint alkalmazások gyűjteményének. Tükröznie kell azt, aki vagy. A Plasma Active megtölti a tableted azokkal az okosságokkal, amelyek támogatják, bármit is csinálsz bármikor a vadiúj, érintésalapú Aktivitásokkal”.

3.6. Ádámtól és Évától: egy kis nevezéktan

A KDE kezdetben a Kool Desktop Environment rövidítése volt, manapság pedig már egy bonyolult és összetett ökoszisztéma összefoglaló neve. A kiadási közlemények már nem KDE-ről szólnak, hanem KDE alkalmazásokról (KDE Applications), KDE Plasma munkaterületekről (KDE Plasma Workspaces) és KDE fejlesztői platformról (KDE Development Platform). A Plasma munkaterület nem sokkal korábban két dolgot jelentett: az asztali és nagyobb kijelzővel szerelt hordozható számítógépekre szánt Plasma munkaasztalt (Plasma Desktop) és a netbookokra szánt Plasma netbookot (Plasma Netbook). Ezekhez csatlakozott harmadikként október 9-én a Plasma Active, amely most még ugyan szintén csak egyetlen eszköztípusra, a tabletekre szánt felület, ez azonban csak a fejlődési folyamat első lépcsőfoka, a cél ennél sokkal nagyobb.

3.7. Plasma Active One

Ahogy fentebb is olvasható, a Plasma Active nem egyetlen eszköztípust céloz meg, a támogatott eszközök skáláját azonban fokozatosan bővítik, első körben csak Intel-alapú tableteket támogat (WeTab, Intel ExoPC), de már az első bejelentés óta bővült a lista az Nvidia Tegra 2 platformjával, és egyesek sikeresen elindították Nokia N9-en is. Maga a Plasma Active nem csak egyszerűen egy felület, amelyet optimalizáltak a hordozható eszközökre, sokkal több annál. A Plasma Active összetett felhasználói élmény, a már meglévő technológiák felhasználásával és továbbfejlesztésével igényekre

szabva. Egyik alapköve az aktivitás, amely régóta ismert és használt a KDE-n belül, első ránézésre csak egy virtuális asztal, második ránézésre azonban sokkal érdekesebbé válik. Lényegében olyan gyűjtemény, amely az adott tevékenységekhez tartozó alkalmazásokat, fájlokat, kisalkalmazásokat fogja össze, legyen szó zenehallgatásról, irodai munkáról vagy képnézegetésről. A Plasma Active-ban ezek állnak a középpontban, néhány előre definiált aktivitást nyújtva, hogy már bekapcsoláskor azonnal használni lehessen az eszközt, a váltás ezek között pedig egyszerű érintésekkel tehető meg.

Manapság az Interneten történik minden, a közösségek és a tartalmak megosztása fontos része az internetező emberek mindennapjainak. A Plasma Active-ban épp ezért azonnal megosztható bármilyen tevékenység a Share-Like-Connect segítségével, a zenehallgatástól kezdve az éppen nézegetett képig e-mailen keresztül vagy más formában. A Share-Like-Connect használatával a létrehozott aktivitásainkat másokkal is megoszthatjuk, vagy éppen másoktól tölthetjük le az általuk készített és megosztott aktivitást.

A Plasma Active több, mint a KDE fejlesztőinek egy kísérlete, hogy minél több helyen lehessenek jelen; a basysKom és az open-slx cégek anyagi vagy fejlesztői támogatással segítették a fejlesztést, az Intel pedig ingyen ExoPC-k adományozásával a fejlesztőknek. A Plasma Active-nak nem célja kész megoldás nyújtása, bár lehetséges utat mutat, de a jövőbeli gyártókra van bízva, hogy a rendelkezésre álló elemekből mit építenek fel. A Linux, KDE és Qt alapoknak köszönhetően alkalmazások széles spektruma érhető el készen, legfeljebb az érintés alapú irányításra kell őket felkészíteni.

Mi várható a következő kiadásokban? Természetesen hibajavítások és stabilitást növelő fejlesztések, de új funkciók is érkeznek. A Plasma Active Two automatikus ajánlási rendszert kap, amely a kapcsolódó dokumentumok és információk összegyűjtésében és aktivitásokhoz rendelésében segít. A Share-Like-Connect képességeit is bővítik, valamint továbbfejlesztett gyűjtemény-megjelenítési, -szűrési és -rendezési képességek várhatók a médiatípusokhoz.

A Plasma Active Three tovább bővíti a támogatott eszközök listáját, például set top boxokkal és handheldekkel, de akár autókba épített számítógépekre is telepíthető lesz. Ebben a kiadásban kevésbé dominálnak az új funkciók, sokkal hangsúlyosabb szerepet kapnak a biztonságot növelő fejlesztések.

4. Összefoglalás

Az informatika világa folyamatosan változik, a szabad szoftveres közösségek pedig igyekeznek lekövetni ezeket a változásokat. Nagyon jó ötletek születnek, és a közösségnek köszönhetően gyorsan meg is valósulhatnak, de hogy mennyire terjednek el, az már nehezebb kérdés. Nagy átalakulás zajlik a számítástechnikában, a szabad szoftveres közösségeknek gyorsan kell reagálniuk, hogy ne maradjanak le. A KDE megindult ebbe az irányba, igyekszik elkapni és meglovagolni a hullámokat. Nehéz következtetéseket levonni, hiszen az eddigiek alapján nagyon ígéretes szoftvereket és rendszereket készítenek, de a gyártókon is múlik, mennyire terjednek el. Reménykedjünk, hogy ha a Linux desktop éve nem is jön el talán soha, a szabad szoftver éve elérkezik.

Hivatkozások

[1] http://labs.qt.nokia.com/2011/05/09/thoughts-about-qt-5/

[2] http://en.wikipedia.org/wiki/Qt_%28framework%29

[3] http://wiki.qt-project.org/The_Qt_Governance_Model

[4] http://aseigo.blogspot.com/2011/08/important-announcement-coming-today-at.html

[5] http://blog.martin-graesslin.com/blog/

[6] http://mogorvamormota.hu/2010/11/08/wayland-es-compiz/

[7] http://mogorvamormota.hu/2010/11/05/wayland/

[8] http://kde.hu/node/333

[9] http://dot.kde.org/2011/10/24/plasma-active-arm

Heurisztikus regex mintaillesztés a FreeBSD-ben a TRE könyvtár segítségével

Kövesdán Gábor <gabor@kovesdan.org>

Kivonat

Tartalomjegyzék

1. Bevezetés **100**

2. Néhány alapvető fogalom **100**
2.1. Mi az a reguláris kifejezés? . . . 100
2.2. Mi az a grep? . . . 100
2.3. A reguláris kifejezések megvalósítása . . . 101

3. Egy ötletes megoldás: GNU grep **101**
3.1. Néhány programozási best practice . . . 101
3.2. Heurisztikus közelítés . . . 101

4. Szövegkeresési algoritmusok **102**

5. Az általános megoldás **102**
5.1. Miért nem működik egy az egyben? . . . 102
5.2. Prefix heurisztika . . . 103
5.3. Fix hosszú heurisztika . . . 103
5.4. Heurisztika tömb . . . 103

6. Konklúzió és távlati kilátások **103**

7. Köszönetnyilvánítás **103**

1. Bevezetés

Az itt bemutatott praktikák a FreeBSD-ben történt fejlesztések során lettek kidolgozva. A cél az elavult regex kód lecserélése volt a C könyvtárban. Fontos követelmény volt az új implementációval kapcsolatban az elfogadható licenc, a POSIX szabványnak való megfelelőség, a hatékonyság, az elfogadható memóriahasználat, illetve a különböző karakterkészletek támogatottsága. A teljesítmény és a memóriahasználat tekintetében felvetődött, hogy a GNU szoftverek determinisztikus automatát használnak pár heurisztikus optimizációval a grep programban, ezért nagyon gyorsak, de bizonyos esetekben nagyon sok memóriát fogyasztanak. Ráadásul a GNU grep program heurisztikákkal kikerüli a regex implementációt, ami nagyban növeli a kód komplexitását, ráadásul az így megvalósított optimizációkból más program nem részesül, csak a grep. Ezen tapasztalatok alapján fogalmazódott meg pár gondolat, hogy hogyan lenne érdemes a FreeBSD új regex implementációját szervezni. Az ötlet egyrészt az volt, hogy az optimizációkat magába a C könyvtárba kellene beépíteni, és nem az azt használó segédprogramokba, így a kód szervezettebb lenne, és minden segédprogram profitálna belőle. Másrészt az tűnt logikusnak, hogy vegyünk egy jól megtervezett nem-determinisztikus automata alapú implementációt, majd azt lássuk el ezekkel a heurisztikus adottságokkal. Ha nagyrészt a hatékonyabb heurisztika algoritmus fut le, és az automata csak kevésszer hívódik, akkor a nem-determinisztikus automata teljesítménybeli hátránya mérséklődik, míg a memóriafelhasználás továbbra is kedvező marad. Habár ezek az ötletek a GNU grepben csírájukban már léteztek, az előzetes vizsgálatok alapján eddig nem jött létre olyan reguláriskifejezés-implementáció, amely heurisztikákkal működött volna. A konferencián megtartott előadás és jelen cikk a fejlesztés tapasztalatait mutatják be. A kiindulópontot a TRE implementáció jelentette, ez lett heurisztikus adottságokkal ellátva.

2. Néhány alapvető fogalom

2.1. Mi az a reguláris kifejezés?

A reguláris kifejezés – röviden regex – olyan szövegminta, amely valamilyen tulajdonságú szöveg ír le, pl. az „ab” karakterrel kezdődő 5 hosszúságú szövegrészeket. Ha egy szövegrész megfelel a mintának, akkor azt mondjuk, hogy illeszkedik a mintára. Az ilyen mintákat a gyakorlatban használhatjuk többféle célra, pl. valamilyen bonyolultabb keresésre, vagy validálási műveletre. Például konkrétan megadhatunk egy olyan mintát, amelyre az e-mail címek illeszkednek, így ellenőrizhető, hogy a beírt e-mail cím helyes formátumú-e. A POSIX szabvány leírja, hogy hogyan épülnek fel ezek a reguláris kifejezések. Kettő változatot is definiál, az egyszerű és az összetett reguláris kifejezéseket. Ezeknek a szintaktikája fellelhető a szabványban, illetve szakkönyvekben is, ezért – illetve terjedelmi okokból is – most részletesen nem ismertetjük a reguláris kifejezéseket.

A POSIX szabvány a reguláris kifejezéseken kívül megköveteli azt is, hogy a C könyvtár támogassa a reguláris kifejezések mintaillesztését, és definiál egy konkrét API-t is, amelyen keresztül a mintaillesztés programozható.

2.2. Mi az a grep?

A grep egy parancssoros segédprogram, amely szöveges fájlokban egy megadott mintára illeszkedő sorokat keres, azaz tulajdonképpen a keresést valósítja meg a fájlokban. Használata nagyon elterjedt, sok feladathoz használható közvetlenül is, illetve szkriptekben is gyakran felbukkan. Sok – a keresést befolyásoló – funkciója van. A POSIX szintén megköveteli a grep program jelenlétét az operációs rendszerben.

2.3. A reguláris kifejezések megvalósítása

A reguláris kifejezések mintaillesztése leggyakrabban egy automatával van megvalósítva. Az automata egy olyan modell, amelynek állapotai vannak, és a bemenet hatására ezen állapotok között vált. Az automatáknak két típusuk van: determinisztikus és nem determinisztikus. Az előbbi mindig egyszerre egy állapotban tartózkodik, míg az utóbbi lehet egyszerre több állapotban is. Általánosságban elmondhatjuk, hogy a determinisztikus automaták gyorsabbak, de több állapottal rendelkeznek, ami nagyobb memóriafelhasználást is jelent. Extrém esetekben az n állapotú nem determinisztikus automata determinisztikus megfelelője 2^n állapottal is rendelkezhet. Ez sokszor már túl nagy memóriafelhasználást jelent. A nem determinisztikus automatáknál nincs ilyen probléma, de ezek nem olyan hatékonyak.

3. Egy ötletes megoldás: GNU grep

A GNU projekt grep implementációja az évek során sok tapasztalatot halmozott fel, amelyek a mintaillesztés megvalósításánál nagyon hasznosnak bizonyulnak. Azonban azt is hozzá kell tennünk, hogy a grep a reguláris kifejezések használatának egy példája, de nem minden alkalmazható általános esetben.

3.1. Néhány programozási best practice

Az alapvető megközelítés, amit a GNU grep alkalmaz, arra épül, hogy egy program akkor lehet igazán gyors, ha a minimálisra korlátozza az elvégzendő feladatokat. Ez jelen esetben több dolgot jelent.

Először is, ahol csak lehet, kerüljük el a másolást. Ne keressünk sortöréseket, és ne másoljuk át külön pufferbe a sorokat. Csak akkor foglalkozzunk a sortörésekkel, ha már találtunk egy illeszkedést, mert elég ezután elkülöníteni az illeszkedő sort. Ez alól egy kivétel, ha meg kell számolnunk a sorokat, és tudnunk kell, hogy melyik sorban volt az egyezés. A sortörések keresése azt is eredményezné, hogy a bemeneti szöveg minden egyes karakterét fel kell dolgoznunk, még ha egyébként erre nem is lenne szükség. Ezért a sortörések vizsgálatát érdemes elhalasztani az utánra, hogy megtaláltuk az illeszkedést.

Másrészt, ha hagyományos NUL-végű karakterláncokat használunk, akkor ugyanarra a problémára jutunk mint az előző esetben: karakterenként kell feldolgoznunk a bemeneti szöveget, hogy megtaláljuk a záró NUL karaktert. Ehelyett egy jobb ötlet, ha számon tartjuk a pufferünk hosszúságát, mert így adott esetben nagyobb ugrásokat is el lehet érni. Sajnos a POSIX API függvényei NUL-végű karakterláncok, tehát el kell térnünk a szabványtól ahhoz, hogy hatékonyak lehessünk.

3.2. Heurisztikus közelítés

A GNU grep a megadott reguláris kifejezésnek, vagy kifejezéseknek veszi a leghosszabb olyan részét, amelynek egy az egyben szerepelnie kell a szövegben, majd a Boyer – Moore illetve a Commentz – Walter algoritmusok egyikével – attól függően, hogy egy vagy több mintánk van – megkeresi az első illeszkedést. Ezek olyan algoritmusok, amelyek fix szövegrészt keresnek hatékonyan egy bemeneti szövegben. Ezután a GNU grep különválasztja az illeszkedő sort, majd ezen a soron a determinisztikus automatát is lefuttatja. Ez a módszer a gyakorlatban nagyon gyors, de látható, hogy ha pl. minden sor illeszkedne, akkor ez teljesítményromlást jelentene, mert akkor az automata és a heurisztika is lefutna minden soron. Ez a megoldás tehát nem mindig hatékony, de a gyakorlatban jól bevált. Az általános megoldásban is erre törekszünk.

4. Szövegkeresési algoritmusok

Mielőtt továbblépnénk az általánosan működő megoldások felé, ejtsünk pár szót a szövegkeresési algoritmusokról, konkrétabban most azokról, amelyek egyetlen mintát keresnek a bemeneti szövegben, mint pl. a Boyer – Moore algoritmus. Ezek az algoritmusok tipikusan hátulról kezdik az összehasonlítást, majd eltérések esetén, az eltérés típusától függően adott karaktert ugranak előre. Kezdetben az algoritmus tehát a mintát a szöveg kezdetéhez igazítja, majd megnézi, hogy az utolsó betű egyezik-e, majd az utolsó előtti stb. egészen addig, amíg a minta első karakteréig el nem érkezünk. Ha eltérés van, akkor pedig nem is megy végig az összehasonlítás, mivel a korábbi karaktereknek nincs jelentősége, hanem tovább csúsztatjuk a mintát és újra kezdődik az összehasonlítás.

A Boyer – Moore algoritmusnál kétféle ugrási módszer van, ezek közül mindig a maximálisat vesszük. Az első a „bad character shift”, ami azt teszi, hogy ha x helyett y betű állt a bemenetben és a mintának az elejében van y betű, akkor azt odaigazítja a megfelelő ugrással, ha pedig nincs ilyen betű, akkor átugorja. A másik módszer a „good suffix shift”, ami pedig úgy működik, hogy ha pl. az eltérés után következő rész szerepel a mintában, de előtte nem olyan betű áll, mint ami épp jött, akkor ehhez igazítja a mintát, mivel ott potenciálisan egyezés lehet, azaz a helyes utótaghoz igazít.

A Boyer – Moore algoritmus egyik változata a Quick Search algoritmus, ami ugyanígy működik, de csak az első módszert használja ugrásra. Ezt könnyebb is kiszámolni, és így egyszerűbbé válik a kód, viszont hatékonyságából nem veszt sokat.

Ennek az utóbbi algoritmusnak az előnye még, hogy könnyen adaptálható úgy, hogy a reguláris kifejezésekben használt „joker” karaktert, a pontot is kezelje, habár az utolsó pont helye limitálja a maximálisan lehetséges ugrást, így romlik a hatékonyság. Sean C. Farley implementált egy ilyen megoldást a freegrep nevű grep változatban.

5. Az általános megoldás

Szóba került korábban, hogy a POSIX API betartásával sajnos nem lehet kihasználni az említett optimizációkat, mivel akkor a feldolgozás karakterenként történik. Szerencsére a TRE implementáció rendelkezik egy alternatív API-val, amely alig tér el a szabványtól, de ezeknek átadhatjuk a karakterláncok hosszát is, így nem kell azt végigolvasni a lezáró NUL karaktert keresve. Ezen kívül a REG_STARTEND nem szabványos flag használata – amely a BSD rendszereken létezett, és jelen munka részeként a TRE-hez is elkészült – esetén sem kell végigolvasni a szöveget, mert a megadott offszetekből kiszámolható a szöveg hossza.

5.1. Miért nem működik egy az egyben?

A grep segédprogramban soronként keressük az egyezést, vagyis az illeszkedő részek nem tartalmazhatnak sortörést. Amikor a leghosszabb fix részt megtaláljuk, akkor ezért, amikor meghívjuk lefuttatjuk az automatát, már csak az előző sortörésig kell visszamennünk, és csak a következő sortörésig kell futtatnunk azt. Általános esetben ez nem működik, mert az egyező részben lehet sortörés is, tehát a sor elejétől kellene kezdenünk a keresést. Azonban ezt az ötletet fenntarthatjuk arra az esetre, amikor a mintát a REG_NEWLINE flaggel dolgozzuk fel, mert ez ugyanezt eredményezi.

Ezen kívül a GNU grep több minta esetén először a Commentz – Walter algoritmussal közelít, a POSIX API pedig egyszerre egy mintával dolgozik. Be lehetne erre is vezetni valamilyen alternatív interfészt, de jelenleg ez nem célja a projektnek.

5.2. Prefix heurisztika

Hogy behatároljuk tehát a potenciális illeszkedés kezdetét, azaz kizárjuk az azt megelőző részeket, kézenfekvő, hogy ne a leghosszabb fix részt, hanem a kezdeti fix részt használjuk. Ha ez a heurisztika találatot eredményez, akkor ennél előbbre már nem kell visszamennünk. Ez csak akkor nem működik, ha már az első karakter is többféle lehet, vagy opcionális.

Az ilyen fajta heurisztikára példa lehet az int[13][62]_t minta esetén az int keresése. De ha a minta u?int[13][62]_t akkor jelen esetben nem tudunk semmit sem tenni, továbbra is az automatát kell használnunk.

5.3. Fix hosszú heurisztika

A korábban tárgyalt általánosítások alapján az is elképzelhető lenne, hogy a nem fix részeket ponttal helyettesítsük. Ezt mindaddig megtehetjük, amíg fix hosszú karaktersorozatot reprezentálnak. Pl. int[13][62]_t helyett kereshetünk int.._t, de int[0-9]*_t esetén ezt már nem tudjuk megtenni, mert a * karakter miatt az adott részen különböző hosszúságú szövegrészek egyaránt illeszkedhetnek. Ez a heurisztika pontosabb, tehát kevesebb „false positive” fordulhat elő, azaz kevesebbszer kellhet futtatni az automatát, de ugyanakkor a pontok lecsökkentik a maximális ugrást. A gyakorlatban dönthető el, hogy melyik módszer tűnik hatásosabbnak.

5.4. Heurisztika tömb

Ennél még egy lépéssel továbbmehetünk úgy, ha megkeressük az összes fix szövegrészt, és ezeket eltároljuk egy tömbben, majd egymás után keressük ezeket a bemenetben. Így ráadásul pontosabb is lesz a heurisztikánk, mert ha egy rövid prefix talál egy „false positive” egyezést, attól még lehet, hogy a további minták kiszűrik azt, így az automata kevesebbszer hívódik meg. Elképzelhető az a megoldás is, hogy a prefix és suffix között csak a leghosszabb köztes mintát vesszük figyelembe, amivel így a legnagyobb ugrások érhetőek el. Ezek a minták tartalmazhatnak pontot, vagy lehetnek teljesen fix minták is.

6. Konklúzió és távlati kilátások

A fejlesztés még mindig folyamatban van, de mostanra már bebizonyosodott, hogy a módszer működőképes. Az eddigi mérések azt mutatták, hogy általában a mohó megközelítés hatékony a gyakorlatban, azaz amikor az éppen legnagyobb lehetséges ugrásokra törekszünk. Sok esetben 40 – 80% közötti teljesítményjavulást is sikerült elérni. Hátra van még sok tesztelés, illetve pár módszer kipróbálása is. Ezen kívül a profiling is segíthet a kritikus részek megtalálásában és további optimizálásában. Jelenleg a kód még nem került be se a FreeBSD-be, se a hivatalos TRE kiadásba. A távlati tervek szerint a FreeBSD mindenképp felhasználja majd a kódot, a TRE eredeti szerzőjével pedig a cikk szerzője egyeztetni fog arról, hogy hogyan lenne lehetséges beolvasztani a kódot az eredeti kiadásba, hogy a szélesebb felhasználói tábor is profitáljon ebből a fejlesztésből.

7. Köszönetnyilvánítás

A cikk szerzője köszönetet kíván mondani Mike Hartelnek, a GNU grep eredeti szerzőjének, aki megosztotta azon tapasztalatait, amelyeket a GNU grep fejlesztése során szerzett, Sean C. Farleynek, akinek a freegrepben alkalmazott ötletei kiindulási pontként szolgálhattak, Ville Laurikarinak, amiért egy ilyet jól átlátható és jól átgondolt szoftvert készített, mint a TRE, valamint Xin Linek,

aki a szerző FreeBSD fejlesztői közösségbe való beilleszkedését segítette, mint mentor. Szintén köszönet illeti azokat, akik bármi módon, ötlettel, teszteléssel, hibajavítással hozzájárultak az eddig fejlesztésekhez.

Nyílt kormányzat, „zárt ablakok”?

Dr. László Gábor

Tartalomjegyzék

1. Bevezetés **106**

2. A nyílt fejlesztői modell hatásai **106**

3. A nyílt kormányzat szintjei **107**
3.1. Szellemi közjavak – Adatvagyon 107
3.2. Az adatvagyon hozzáférhetősége és hosszú távú megőrzése 107
3.3. A nyílt szabvány, mint megoldás 108

4. Jogszabályi környezet **109**

5. Nyílt Kormányzat Együttműködés **109**

6. Zárszó helyett **110**

1. Bevezetés

„Az e-kormányzat következő generációjának két követelménye van: az interoperabilitás és az átláthatóság. Ez a kettő tulajdonság az erőssége a nyílt forráskódú szoftvereknek.”

Michel Sapin (2001) francia közigazgatási miniszter

A nyílt kormányzat kérdésköre, hasonlatosan a nyílt szoftverekhez, több kérdést vet fel, mint amennyit egy előadás keretén belül meg lehet válaszolni. A nyílt fejlesztői modell, párhuzamosan az IKT eszközök fejlődésével és azok konvergenciájával hatást gyakorol(t) mindennapi életünkre is. A Web2 modellje átalakította a kultúrát és az állampolgárok politikai részvételét is a döntéshozatal folyamatában. A nyílt forráskód modellje nemcsak a szoftvervilágban és a tudásintenzív gazdasági életben hozott jelentős változásokat, hanem a politikai életben is kezdi éreztetni közvetlen hatását.

Az előadás célja – a teljesség és a definitív megfogalmazások nélkül – a nyílt kormányzat komplex jelenségkörének, hátterének, fejlődési irányainak bemutatása.

2. A nyílt fejlesztői modell hatásai

„A demokrácia az első nyílt forráskódú alkalmazás.”

Phil Windley

Milyen problémákat oldhatunk meg a nyílt együttműködés modelljével? Milyen módon befolyásolja a tanulásról, a munkáról, a kormányzatról/kormányzati munkáról alkotott elképzeléseinket? A nyílt forráskód fogalma a szoftver forráskódjára, az együttműködés modellje pedig arra utalva jött létre, hogy hogyan fejlesztik ezeket a szoftvereket. A nyílt forráskód alapelvei: *nyíltság, átláthatóság, együttműködés, változatosság*. A nyílt forrás több mint szoftverfejlesztési modell, leírja napjaink kultúrájának jellemzőit, az ötletek és az erőfeszítések megosztását, együttműködést és nem elszigetelődést jelent. A jelenség azonban már régen túlmutat a technológián, a szoftver forráskódján. Hatása exponenciálisan növekszik az élet minden területén.

A nyílt modell legújabb alakváltozatát a tömeges részvétel és az együttműködés jellemzi, túllépve a fejlesztői közösségeken. A fogalmat „wikinomics” – a magyar nyelvű fordítása „wikinómia” – Don Tapscott 2006-ban megjelent könyve kapcsán vezette be. A technológiai forradalom, a demográfiai, a társadalmi és a gazdasági változások együttesét hívja így. A wikinómia alapelvei a nyíltság, az egyenlőség, a részvétel és a globális gondolkodásmód [Tapscott 2006]. A tömeges együttműködésre épülő modell megváltoztatja a gazdaságot, a társas kapcsolatokat, az információáramlást, a tudásmegosztást és még a kormányzati szervezeteken belül is érezteti hatását. A közösségi oldalak és a Wikipedia megjelenése és gyors növekedése, valamint a „wiki”, mint technológia elterjedése és intenzív használata jellemzi a Web2-ként is említett korszakot. [Kelty 2008] szerint a végbemenő változásokat a szabad szoftverek megjelenésének és hatásainak szemszögéből lehet megérteni.

Az együttműködés új korszaka, a 2.0-val jellemzett korszak nem igazán technológiai, hanem szemléletbeli változást jelent. A Web 2-es alkalmazások analógiájára sok terület változásait jellemezték az 1.0-s verzióról a 2.0-ás verzióra történő átállással. A változások oka az internet, az internetes szolgáltatások változásában keresendő.

A „nyílt forráskód” modell a szoftvertechnológián túlmutatva demokratizálódási hatásokat is kivált. Ez az effektus a fejlett nyugati világban is érezteti hatását csakúgy, mint az ilyen típusú szoftvereket előnyben részesítő „Globális Dél” államai esetében. Mindkét esetben más-más típusú változásokat segítve elő.

A legismertebb és egyik legjelentősebb kezdeményezés a Barack Obama által indított Nyílt Kormány Kezdeményezés [Open Government Initiative 2009] jelszavai: *átláthatóság, részvétel, együttműködés*.

3. A nyílt kormányzat szintjei

> *„A szabadságszerető ember – minden felismert közérdek ügyében kezdeményezőleg lép fel, minden közérdekű szövetkezésben vagy mozgalomban tehetsége szerint munkájával és adományával részt vesz, és igyekszik azt győzelemre segíteni, tisztában lévén azzal, hogy a közügyek elhanyagoltsága, vagy méltatlan emberek kezébe való kerülése egyedül a tisztességes emberek kezdeményezésének hiánya és közéleti bátortalansága miatt történhetik."*
>
> Bibó István: A szabadságszerető ember politikai tízparancsolata

A nyílt kormányzat fogalma alatt kezdetben a kormányzati és az állami hivatalok döntési folyamatainak átláthatóságát (nyíltságát) és adatainak hozzáférhetőségét értették. [Perrit 1997]

A nyílt kormányzat fő témakörei közt szerepel az adatok hozzáférhetősége, a hosszú távú megőrzés és elérés, az adatvagyon hasznosulása, továbbá az állampolgárok politikai részvétele a döntéselőkészítésben, a döntéshozatal kontrollálásában.

3.1. Szellemi közjavak – Adatvagyon

A világháló nagyszerű új lehetőségeket teremtett a tudásmegosztáshoz, a kultúra terjedésének új formáihoz, az alkotások megosztásához. Azonban az internet megjelenésével és ezekkel a változásokkal a jog nem tudott lépést tartani. Lessig (2005) Szabad Kultúra című művében alaposan körüljárja a szellemi tulajdon, a szerzői jog kialakulását és szerepét, a jelenlegi rendszer visszásságait és azokat a problémákat, amelyeket az internet idézett elő ezen a területen, valamint elemzi a szellemi közjavakat is. [Lessig 2005]

A közszektor az államháztartás, a közszolgáltató vállalatok és a non-profit intézmények összessége. Funkciója a kollektív javak (közjavak) és a közszolgáltatás biztosítása. A szellemi közjavak a közszféra számára különös jelentőséggel bírnak.[Rátai, B. – Szemes, B. (2008)] A közszférán belül is átláthatóbb és olcsóbb működéshez vezethetnek, amelyek magukba foglalhatják a nyílt forráskódú szoftvereket is. A szellemi közjavak tartománya azonban önállóan is szerteágazó terület. Az előadás két kiemelt területtel foglalkozik, a technológiai megjelenítéssel, hosszú távú megőrzéssel, valamint a jogszabályi környezettel.

3.2. Az adatvagyon hozzáférhetősége és hosszú távú megőrzése

Az UNESCO megfogalmazása szerint: „A világ digitális örökségét a végleges eltűnés veszélye fenyegeti. Ehhez hozzájárul az azt tartalmazó hardware és software gyors elavulása, bizonytalanság a forrásokkal, felelősségekkel, és a fenntartás és megőrzés módszereivel kapcsolatban, valamint a támogató jogi szabályozás hiánya. A hozzáállásbeli változások elmaradtak a technológiai változások mögött. A digitális fejlődés túl gyorsan és a kormányok és intézmények számára túl költséges módon ment végbe ahhoz, hogy a megfelelő időben lehessen megőrzési stratégiákat életbe léptetni. Az

örökség gazdasági, társadalmi, szellemi és kulturális potenciálját – a jövő építő elemeit – érintő fenyegetéseket még mindig nem tudták teljesen megragadni." [Charta a digitális örökség védelméről,, 3. Cikkely – Az eltűnés veszélye]

3.3. A nyílt szabvány, mint megoldás

A nyílt szabványok alkalmazása fontos szerepet kap a kormányzatok és kormányzati szervezetek által közzétett információk megjelenítése és közzététele terén. Vannak kormányzatok, amelyek portálokat (például data.gov) hoztak létre, hogy az adatok könnyebben megtalálhatóak legyenek. A W3C (World Wide Web Consortium) segítségképpen egy útmutatót jelentetett meg, amelyben szereplő egyszerű lépések kiemelik a szabványok és módszertanok használatát, ezáltal is támogatva a kormányzati adatok publikálását és a kormányzati információk innovatív, nagyobb üzleti és közösségi hasznosítását és újrahasznosítását. Ezeknek az adatoknak a megosztása nagyobb átláthatóságot és hatékonyabb közszolgáltatásokat tesz lehetővé. [Publishing Open Government Data]

A Weben történő megosztáson túlmutatóan az informatika „rövid távú" gondolkodására jól világít rá Erik Kriss, Massachusetts állam közigazgatásért és pénzügyekért felelős miniszterének találó megjegyzése, miszerint „az egyik központi kérdés az, hogy miképp biztosítsuk a közadatok operációs rendszertől és alkalmazásoktól való függetlenségét hosszú időtávlatban. Az információs technológia területén a hosszú táv úgy 18 hónap, a kormányzatban ez körülbelül 300 évet jelent, tehát egy kissé különböző távlatban gondolkodunk". [Informal comments on Open Formats n.d.]

A nyílt szabvány definíciójára számtalan megközelítés létezik. A Nemzetközi Távközlési Egyesület Szabványosítási testülete (ITU-T)[1] a nyílt szabványokat olyan szabványként definiálja, amelyek a nagyközönség számára elérhetők, és a kifejlesztésük (vagy az elfogadásuk) és karbantartásuk együttműködésen alapuló, átlátható közmegegyezéses eljárással történik, amelyben valamennyi érintett szabadon részt vehet. A nyílt szabványok, érinthetnek tulajdonosi szabadalmakat. A definíció sok vitát generált ezen szabadalmak licencelésével kapcsolatban. Az elfogadható, méltányos és diszkriminációmentes (reasonable and non-discriminatory – RAND) feltételek alatt rendelkezésre bocsátott szabványok a nyílt forráskódú közösségek számára – általában – nem fogadhatók el nyílt szabványként, de a legtöbb szabványügyi szervezet és testület az ezen feltételeket alkalmazó szabványokat is nyílt szabványként kezeli. [Hoe 2006]

A nyílt szabványra – a definíciók és a különböző megközelítések ötvözésével – az alábbi meghatározást tehetjük: Egy szabványt akkor tekinthetünk teljesen nyílt szabványnak, ha teljesíti az alábbi kitételeket:

- Szabadon megismerhető és használható;
- Szabadon implementálható, a fejlesztése nyílt folyamat eredménye, bárki részt vehet benne;
- Független a gyártóktól és szállítóktól;
- Jogdíjmentes.

A nyílt szabvány és a nyílt forráskód kapcsolatában a nyílt forráskód a nyílt szabványok érvényesülését segíti.

Magyarországon a Parlament 2009. december 14-én 197 igen, 1 nem, 146 tartózkodás arányban elfogadta azt a törvényjavaslatot, amely a Nyílt Szabvány Szövetség által javasolt nyílt szabványos törvénymódosítást tartalmazta[2]. (2009. évi LX. törvény az elektronikus közszolgáltatásról) A nyílt kormányzás azonban nem csak a technológiai feltételeket foglalja magába.

[1] Definition of „Open Standards", `http://www.itu.int/en/ITU-T/ipr/Pages/open.aspx`

[2] A törvénymódosítás kulisszatitkai, `http://nyissz.hu/blog/a-torvenymodositas-kulisszatitkai/`

4. Jogszabályi környezet

A jogszabályi környezet is többrétű. Egyrészt, magát az információszolgáltatást foglalja magában, másrészt, a keletkezett adatvagyon szerzői jogi szabályozását, főként annak üzleti célú további hasznosítása esetén.

Az Eötvös Károly Intézet Az elektronikus információszabadság jogi garanciáinak felülvizsgálata című tanulmányának megállapítása szerint, *„Magyarország példás információszabadság-szabályokkal rendelkezik*, szabályozási megoldásai sok más ország számára szolgálnak mintaként. A szabályozás korszerűsége többek között abban áll, hogy nem az akták, még csak nem is a dokumentumok, hanem az azokban szereplő adatok nyilvánosságára koncentrál, függetlenül a hordozójuktól vagy az adatok önálló vagy gyűjteményes jellegétől". [. . .] „Az elektronikus információszabadság alapvetően jó szabályai azonban mégsem érvényesülnek megfelelően. Több felmérés zárult a közelmúltban olyan eredménnyel, miszerint a közzéteendő adatok Magyarországon a gyakorlatban csak a törvény szerint nyilvánosak, valójában nem megismerhetők". [. . .] „Az információszabadság érvényesülésével kapcsolatos felmérések azt mutatják, hogy a szabályok a hivatalok nyilvánosság-párti hozzáállása nélkül nem sokat érnek".[Eötvös Károly Intézet 2009]

Az elsődlegesen meghatározó maga a kormányzat hozzáállása a kérdéskörhöz. Magyarországon – jóllehet mindkét program foglalkozik a nyílt kormányzat kérdésköreivel – sem a Digitális Megújulás Cselekvési Tervben[3] sem a Magyary Programban[4] nem szerepel a „nyílt kormányzat" nevesítve.

5. Nyílt Kormányzat Együttműködés

Átláthatóság, részvétel, együttműködés a jelszavai az új Nyitott Kormány Direktívának (Open Government Directive), amelyet Barack Obama kampányígéretéhez híven 2009-ben indított az amerikai kormány munkájának átláthatóbbá tételére [Memorandum 2009]. A kezdeményezés nemzetközi kitekintésben a Nyílt Kormányzat Együttműködés programjában csúcsosodott ki, amelyet hivatalosan 2011. szeptember 20-án indítottak útjára 8 alapító ország (Brazília, Indonézia, Mexikó, Norvégia, Fülöp-szigetek, Dél-Afrika, Nagy-Britannia és az USA) részvételével, amely országok jóváhagyták a Nyílt Kormányzat Nyilatkozatot, és bejelentették, országaik cselekvési terveit. További 38 kormányzat jelezte elkötelezettségét az együttműködéshez[5].

Magyarország a honlap tanulsága szerint még nem jelezte csatlakozási szándékát az együttműködéshez, amelyre civil szervezetek fel is hívták a Kormány figyelmét 2011. szeptember 29-én kelt nyílt levelükben. [Nyílt levél Navracsics Tibor részére]

Annak érzékeltetésére, hogy a nyílt kormányzathoz is hosszú és szisztematikus út vezet, érdemes egy rövid kitekintést tenni annak az országnak a példájára, amely hosszú éveken keresztül ez e-kormányzat éllovasa volt és amely a bejelentés napján – 2011. szeptember 20-án – csatlakozott a Nyílt Kormányzat Együttműködéshez. A kanadai kormányzat az 1960-as és 70-es években felismerte a lakosság jogát a nyilvános kormányzati adatokhoz való hozzáféréshez. Abban a kérdésben is egyetértés mutatkozott, hogy az adatbázisok előállításába és működtetésébe be kell vonni a civil szektort, ügyelve arra, hogy a *kormányzat tulajdonában és ellenőrzése alatt* maradjanak az adatbázisok. 1993 szeptemberében az Industry and Science Canada közzétette, hogy elindított egy online dokumentációs adatbázist, amelyben minden dokumentum angol és francia nyelven is elérhető ASCII

[3] Digitális Megújulás Cselekvési Terv 2010 – 2014, `http://www.kormany.hu/download/6/4f/00000/Digitalis_Megujulas_Cselekvesi_Terv.pdf` Letöltve: 2011. október 20.

[4] Magyary Közigazgatás Fejlesztési Program, `http://www.kormany.hu/download/1/6d/40000/Magyary Kozigazgatas fejlesztesi Program.pdf` Letöltve: 2011. október 20.

[5] A programhoz kapcsolódó országok listája megtalálható az együttműködés honlapján: `http://www.opengovpartnership.org/countries` Letöltve: 2011. október 20.

formátumban, FTP protokollon, vagy Listserve-n keresztül. A kapcsolódások száma az induláskori 150 kapcsolat/hétről következő év januárjára 7 000-re emelkedett. Ezt látva a kezdeti 500 dokumentum helyett az egész adatbázist könnyen hozzáférhetővé, és teljes mértékben kereshetővé tették januárban. Az Internet hasznosságát egyre több kormányzati szerv és hivatal ismerte fel. 1994-ben megszaporodott azoknak a teljes szövegű dokumentumoknak a száma, amelyek elérhetővé váltak az FTP, Gopher, WAIS, illetve az Internet segítségével. 1994 januárjában a kormány az informatikai vezetőik részére egy útmutatót bocsátott ki az információs vagyon (ezzel együtt a technológiák) felülvizsgálatára. Ez felölelte az információ egész életciklusát (keletkezés, rendszerezés, tárolás, megsemmisítés stb.). 1994 áprilisában a Pénzügyi Bizottság egy vitaanyagot bocsátott ki, amelynek célja a kormányzati szolgáltatások újragondolása volt az informatikai eszközök használatával kapcsolatosan. Ezt széles körben terjesztették, *várva a visszajelzéseket a kormányhivatalokban dolgozóktól éppúgy, mint a lakosságtól*. A „látomás" egy egységes kormányzati információs infrastruktúra volt, amely felgyorsítja az adatáramlást, csökkenti a duplikációt, jobban és költséghatékonyabban lássa el mind a kormányzati munka, mind a lakosság szolgálatát. A kanadai kormány példaértékű modell felhasználóvá vált, ezzel is elősegítve az új technológiák mindennapi bevezetését az országban. 1995 decemberében indították útjára a kanadai kormányportált a www.canada.gc.ca címen, az állampolgárok jobb kiszolgálása érdekében. Az Industry Canada stratégiai gyűjtőoldalán1997-ben több, mint 650 000 dokumentum volt online elérhető[6].

2011-ben Open Government néven külön gyűjtőoldalt hoztak létre a www.open.gc.ca címen, amelyen a nyílt kormányzattal kapcsolatos iniciatívák és információk találhatóak meg.

6. Zárszó helyett

Amint azt az előadás bemutatta, a nyílt kormányzat kérdésköre, hasonlóan a nyílt szoftverekhez összetett jelenségkör. Hogy nyílt kormányzattal, vagy zárt ablakokkal találkozunk, az nagyrészt a kormányoktól függ, azonban az állampolgárok is sokat tehetnek annak érdekében, hogy kitáruljanak az ablakok...

Hivatkozások

[Tapscott 2006] Tapscott, D. – Williams, A. D. (2006). Wikinomics. Portfolio. USA

[Kelty 2008] Kelty, (2008). Two bits. The Cultural Significance of Free Software. Two Bits könyv letölthető a webről különböző formátumokban. a forrás: `http://twobits.net/` Letöltve: 2009. február 1.

[Open Government Initiative 2009] Open Government Initiative, `http://www.whitehouse.gov/open/`

[Perrit 1997] Perritt, H. H. Jr. (1997). Open Government. Government Information Quarterly, 14. pp. 397-406.

[Lessig 2005] Lessig, L. (2005). Szabad kultúra – A kreativitás természete és jövője. Kiskapu Kft. `http://www.szabadkultura.hu/`. Letöltve: 2005. október 17.

[Rátai, B. – Szemes, B. (2008)] Rátai, B. – Szemes, B. (2008). Szellemi közjavak: a nyílt forrású szellemi alkotások jövője. Információs Társadalom folyóirat. 2008/1. szám. Infonia. Budapest.

[6]The Internet in Canada – A Background paper prepared by Catherine Peters, Industry Canada for the Internet Steering Committee of the Information Highway Advisory Council, April 1997, p8.

[Charta a digitális örökség védelméről,] Charta a digitális örökség védelméről, UNESCO, http://www.unesco.hu/archivum-2009-vegeig/kommunikacio-informacio/charta-digitalis-orokseg Letöltve: 2011. szeptember 12.

[Publishing Open Government Data] Publishing Open Government Data. W3C Working Draft 8 September 2009, http://www.w3.org/TR/2009/WD-gov-data-20090908/ Letöltve: 2011. szeptember 21.

[Informal comments on Open Formats n.d.] Informal comments on Open Formats n.d., http://www.mass.gov/eoaf/open_formats_comments.html. Letöltve: 2005. december 21.

[Hoe 2006] Hoe, N. S. (2006). Free / Open Source Software. Open Standards. International Open Source Network. Elsevier.
Forrás: http://www.iosn.net/government/foss-government-primer/foss-govt-policy.pdf. Letöltve: 2005. október 12.

[Eötvös Károly Intézet 2009] Eötvös Károly Intézet (2009): Az elektronikus információszabadság jogi garanciáinak felülvizsgálata
http://www.ekint.org/ekint_files/File/tanulmanyok/ekint_tanulmany_efoi_fv_nhit.pdf
Letöltve: 2009. november 22.

[Memorandum 2009] Memorandum for the heads of executive departments and agencies,
http://www.whitehouse.gov/omb/assets/memoranda_2010/m10-06.pdf
Letöltve: 2010. december 3.

[Nyílt levél Navracsics Tibor részére] Nyílt levél Navracsics Tibor részére,
http://tasz.hu/files/tasz/imce/2011/ogp_level_kim_miniszter.pdf
Letöltve: 2011. október 21.

LibreOffice – úton a DTP felé

Németh László <nemeth@numbertext.org>

Tartalomjegyzék

1. Bevezetés **114**

2. Mit nevezünk kiadványszerkesztőnek? **114**

3. Nem hagyományos kiadványszerkesztés? **114**

4. Dokumentumszerkesztőből kiadványszerkesztő? **115**

5. Kiadványszerkesztés LibreOffice Writer szövegszerkesztővel **115**

6. Magyar nyelvi és tipográfiai támogatás **115**

7. Mitől „fapados” kiadványszerkesztő a LibreOffice? **117**

8. Melyiket válasszuk? **117**

9. Példa az EPS-támogatásra **119**

10. Példa az OpenDocument használatára **120**
10.1. ODFpy 120
10.2. Példa ODF állomány létrehozására 120
10.3. Betűkészletek beállítása 121
10.4. Címsor beállítása 121
10.5. Szövegkeretek beillesztése 121
10.6. További példák 122

11. Hogyan tovább? **122**

12. Hivatkozások **122**

1. Bevezetés

Fernand Vanrie, a belga PMG kiadó főmunkatársa komoly problémával nézett szembe. A szakmai magazinokat kiadó cég növekedésnek indult, de az újabb és újabb drága kiadványszerkesztő programok beszerzése túlzásnak tűnt az elvégzendő feladathoz képest. Ekkor került képbe az ingyenes OpenOffice.org, a LibreOffice elődje. Némi próbálkozás után alkalmasnak is tűnt a előzetes tördelésre, vagyis a sok kép és szöveg elhelyezésére, levéve ezt munkát a professzionális kiadványszerkesztő válláról.

Legalábbis ez volt a kiindulási terv, de kiderült, hogy a kidolgozott munkafolyamatban nincs szükség kereskedelmi kiadványszerkesztőre. A kiadó azóta is évi mintegy 8000 színes, képes magazinoldalt készít az OpenOffice.org-gal, ami jó példája az OpenOffice.org, illetve a LibreOffice rejtett kiadványszerkesztő képességeinek. Ilyen, a belga kiadó esetében döntő képesség, hogy az OpenOffice.org/LibreOffice kezeli az EPS (Encapsulated PostScript) képállományokat, így a reklámügynökségek által készített – bittérképes képeket, vektoros feliratokat is tartalmazó – professzionális hirdetési anyagok eredeti nyomdai minőségben kerülhetnek be a magazinokba, nincs szükség ezek előzetes bittérképes átalakítására.

Összevethető-e a LibreOffice a kereskedelmi kiadványszerkesztőkkel? Ha igen, melyek a LibreOffice kiemelkedő képességei, amellyel versenyképes termékként léphet fel nemcsak a szövegszerkesztők (dokumentumszerkesztők), hanem a kiadványszerkesztők világában is? Ki tud-e lépni a LibreOffice a „fapados” kiadványszerkesztő kategóriából, és ha igen, hogyan?

2. Mit nevezünk kiadványszerkesztőnek?

Nyomdai vagy közel nyomdai minőségű kiadványok készítésére szolgáló, grafikus felületű személyi számítógépes alkalmazásokat, amelyek a kiadványok többé-kevésbé alakhű[1] szerkeszthető előnézetét mutatják. Az asztali, vagyis személyi számítógépes kiadványszerkesztés (*desktop publishing* – DTP) az Apple Macintosh rendszerek egyik sikerének titka volt. Közel nyomdai minőségű kiadványok készítését tették lehetővé a PostScript lapleíró nyelvre és az azt közvetlenül értelmező, jó minőségű (300 dpi-s) LaserWriter nyomtatóra támaszkodva 1985-ben. (A technológiának elválaszthatatlan része volt a DTP sikerében.) Idővel nemcsak a különböző vektoros betűtípusokkal és tipográfiai finomságokra is ügyelő szövegtördeléssel készülő könyvek és egyéb kiadványok tartoztak a DTP hatáskörébe, hanem a – jelentős részben a DTP rendszerek fejlődésének köszönhetően egyre összetettebb szöveg- és képelrendezését mutató – színes magazinok is. A hagyományos értelemben vett DTP ezen a területen éri el a csúcsát, még ha a végtermék nyomdai-tipográfiai színvonala erősen szakemberfüggő maradt ma is, hiszen a kiadványkészítés nem nélkülözi az iparművészeti ismereteket és készséget.

3. Nem hagyományos kiadványszerkesztés?

A hagyományos DTP programok nem fednek le minden nyomdai igényt, nagyobb kiadványokhoz az Adobe már a Framemakert ajánlja. Telefonkönyvek, lexikonok, napilapok kiadásában is specializált nyomdai rendszerek (pl. Datalogics DL Composer, Miles 33, KyTek Autopage QuarkXPress bővítmény, PTC APP, XPP) segítenek, nemcsak a matematikai-tudományos folyóiratok szedésben egyeduralkodó a TEX és változatainak használata. Ezekben a rendszerekben közös, hogy esetenként a WYSIWYG rovására (ezeket többé-kevésbé mellőzve) nagyobb automatizálási lehetőséget biztosítanak, többnyire az XML-re támaszkodva. A következőkben a nagyobb kiadványok félig-meddig

[1] WYSIWYG – *What you see is what you get*, magyarítva ALAKHŰ – *Azt látod, amit kapsz, hűen*.

automatizált szedésére mutatunk példát olyan ODFPy Python kódrészletekkel, amelyekkel a teljes Biblia került kiszedésre (ezer oldalnyi szöveg 30 ezer lapszéli kereszthivatkozással). Előtte még megemlítjük azonban a LibreOffice hagyományos asztali kiadványszerkesztésre jellemző, például a nyílt forráskódú Scribus program által is megcélzott képességeit.

4. Dokumentumszerkesztőből kiadványszerkesztő?

Ahogy a bevezetőben már említésre került, ma már egy olyan dokumentumszerkesztő, mint a LibreOffice is be tudja tölteni egy kiadványszerkesztő szerepét, természetesen képességei és a feladat függvényében. A mai dokumentumszerkesztők fokozatosan veszik át a kiadványszerkesztők képességeit: grafikus felhasználói felület; vektorgrafikus betűkészletek, a betűk egalizálása, komplex szövegelhelyezés (hasábok, összefűzhető szövegkeretek), vektorgrafikus képformátumok támogatása, fejlett betűtechnológia, PDF-kimenet stb.

A magyar FSF.hu Alapítvány támogatásával megvalósított fejlesztéseknek köszönhetően a LibreOffice olyan tipográfiai lehetőségeket nyújt alapváltozatában is, amelyek eddig csak a nyomdára és a DTP rendszerekre voltak jellemzők, vagy még azokra sem: ilyen a valódi kiskapitálisok, ligatúrák (köztük a magyar *fj, ffj, gy, gf, gfi, gfö, gj* kurzív ligatúrák), ugráló számok, valódi nagy és kis betűfokozat, magyar csillagos lábjegyzetszámozás stb. támogatása.

Fontos kiemelni a piacvezető kiadványszerkesztők egyik célkitűzését, a munkavégzés hatékonyságának növelését. Kezdve a billentyűkombinációktól a mesteroldalakon át a GREP stílusokig számos ponton biztosítják a mai DTP programok a kiadványszerkesztési feladatok automatizálását. Ez egyáltalán nem áll távol a LibreOffice-tól, amely programozhatóságával (makrórögzítő, UNO programozási interfész, Basic fejlesztői környezet, beépített Python interpretált nyelv, Java és C++ programozási keretrendszerek, kiterjesztéskezelő, beépített és bővíthető XML-szűrők) és alapértelmezett, nyílt szabványú, XML-alapú fájlformátumával (OpenDocument) kiemelkedik a dokumentumszerkesztők és a kiadványszerkesztők között is.

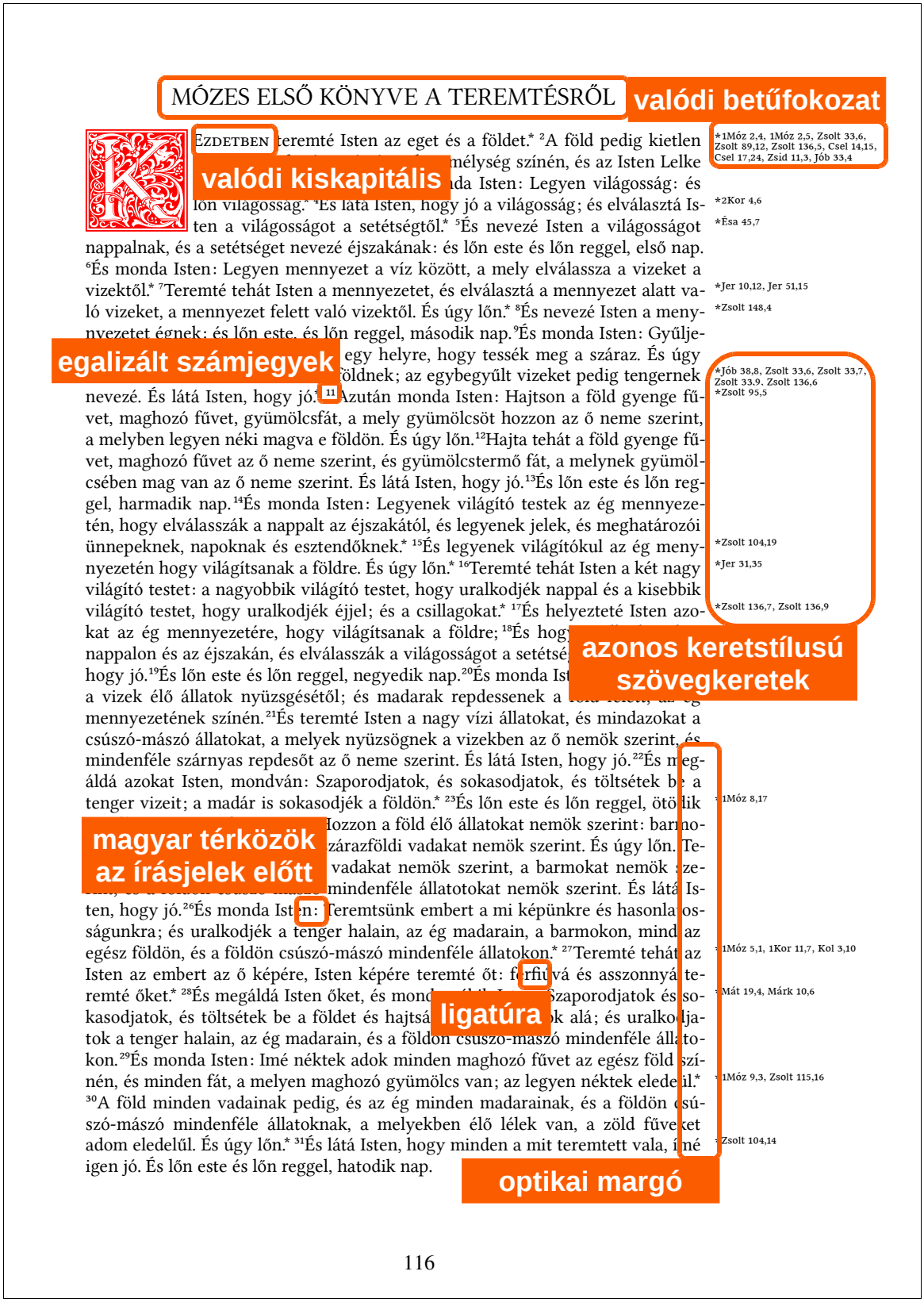
5. Kiadványszerkesztés LibreOffice Writer szövegszerkesztővel

A http://www.numbertext.org/libreoffice címen elérhető magyar nyelvű jegyzet bevezetést nyújt a LibreOffice Writer és eszközeinek, a betű- és bekezdésformázásnak, stílusoknak, sablonoknak, sőt programozási felületének használatába.

6. Magyar nyelvi és tipográfiai támogatás

A LibreOffice kifejezetten kiadványszerkesztőkre jellemző tulajdonságai (például fejlett betűtechnológia, interaktív PDF-elemek támogatása) kiegészül a kiemelkedő magyar nyelvi támogatással: az elválasztás és a helyesírás-ellenőrzés több szinten szavatolja a kiadványaink minőségét. A magyar névelő automatikus kiválasztása a tudományos kiadványok készítését egyszerűsíti le. A Graphite betűtechnológia nemcsak a piacvezető OpenType versenytársa, hanem annál nagyobb rugalmasságot biztosít GDL programnyelvének és gyártófüggetlen megoldásainak köszönhetően. Látványos példája ennek a LibreOffice Linux Libertine G és Biolinum G betűkészleteinek alapértelmezett automatikus ezrestagolása, szóköz helyett a megfelelő keskeny szóközzel. Számos olyan magyar tipográfiai sajátosságot ismer a LibreOffice, amelyeket a honosított kereskedelmi kiadványszerkesztők sem: ilyenek a már említett speciális kurzív ligatúrák, a magyar írásjelekre jellemző térköz az írásjelek előtt, a magyar lábjegyzet-csillagozás, illetve a magyar ékezetes betűk egalizálása (ami nemcsak a Times New Roman betűtípusból, hanem számos egyéb kereskedelmi betűkészletből is hiányzik).

1. ábra. Példa a LibreOffice alapértelmezett DTP lehetőségeire



A LibreOffice szabad szoftver, ami jelentősen hozzájárult ahhoz, hogy a magyar kiadványszerkesztési igényeknek megfelelően lehetett bővíteni. Ez a már említetteken kívül pl. nemcsak a kettőzött többjegyű mássalhangzók automatikus elválasztását tette lehetővé, hanem olyan elválasztási finomságokat is, mint a kötőjelet tartalmazó szavaknál a nagyobb (három) betű távolságra történő elválasztást (alapesetben nincs ilyen elválasztás: helyesírás-elllenőrzés, csak hellyeslírás|ellen|őr|zés), vagy az önmagában maradó fi ligatúrák automatikus cseréjét f (keskenyebb variáns) és i betűkre.

7. Mitől „fapados” kiadványszerkesztő a LibreOffice?

Jó pár olyan alapvető funkció hiányzik belőle, ami a mai piacvezető kiadványszerkesztőkben megtalálható, ilyen például a rétegek kezelése. A színkezelés vagy nyomdai vágás és jelöléseinek hiányát orvosolni lehet a kimeneti PDF vagy PostScript állomány utólagos feldolgozásával. Több képesség meglehetősen rejtve van, például a mesteroldalak megfeleltethetők a LibreOffice Writer oldalstílusainak, ahol ha azt szeretnénk, hogy keretek, képek ismétlődjenek minden ilyen stílusú oldalon, akkor az oldal(stílus) fejlécéhez vagy láblécéhez kell kötnünk ezeket (ettől még az oldal tetszőleges részére pozicionálhatók). Az oldalak hátterét a margók megőrzésével szintén kezelhetjük kerettel, de ez a rétegek hiánya miatt igencsak kényelmetlenné teszi a szövegszerkesztést a LibreOffice-ban (hacsak a szöveg nem keretben foglal helyet, mivel a keretek takarási sorrendje beállítható), ezért a Kiadványszerkesztés LibreOffice Writer szövegszerkesztővel jegyzet egy másik megoldást javasol, a margó nullára állítását, és belső margókkal való helyettesítését.

Ami miatt mégis jó szívvel ajánlható a LibreOffice már a hagyományos kiadványszerkesztési funkciókra, az a fejlett és kiemelt magyar tipográfiai támogatással bíró betűtechnológia (amiben megelőzi a Scribust), továbbá a költséghatékonyság és a már ismerős felület. Ha kis példányszámban, digitális nyomdában készítjük el kiadványainkat, akkor a LibreOffice-ból mentett PDF-ek utólagos feldolgozására sincs szükség (esetleg nagyon nagy felbontású képek beillesztése esetén a PDF-exportálás beállításai között válasszuk ki a Képfelbontás csökkentése opcióban a nyomda számára elegendő 300 dpi-s értéket).

8. Melyiket válasszuk?

A következő táblázat segítséget nyújt a LibreOffice és az egyéb kiadványszerkesztésre is használt piacvezető, illetve piacvezető nyílt forráskódú szoftverek összehasonlításában.

A Scribus brosúrák, magazinoldalak készítésére szánt nyílt forráskódú kiadványszerkesztő, de nemcsak a magyar elválasztási és egyéb sajátosságokat, vagy a nagy kiadványok szedéséhez szükséges tulajdonságokat sem támogatja még, hanem a minőségi kiadványszerkesztés szempontjából alapvető, pl. ligatúrák, ugráló számok, valódi kiskapitálisok egyszerű használatához szükséges betűtulajdonságokat sem, ellentétben a LibreOffice-szal. Viszont a kiadványtípusok, rétegkezelés és nyomdai előkészítés támogatása miatt a kisebb képes kiadványok elkészítésére sokkal inkább alkalmas, mint a dokumentumszerkesztők.

Az Adobe InDesign egyike a piacvezető kiadványszerkesztőknek, számos innovatív megoldással. Legfrissebb változatának egyik újdonsága a Hunspell szótárak támogatása, így a magyar fejlesztésű, az FSF.hu Alapítvány támogatásával is fejlesztett nyílt forráskódú Hunspell helyesírásellenőrző és magyar helyesírási szótára jó szolgálatot tehet ezentúl nemcsak a LibreOffice, hanem az InDesign felhasználóinak is. A magyar elválasztás és tipográfia vonatkozásában viszont még elmarad a LibreOffice-tól.

A Microsoft Office a piacvezető dokumentumszerkesztő, a feltörekvő LibreOffice legfőbb vetélytársa. Fejlett betűtechnológia, stílusok, magyar nyelvi és tipográfiai sajátosságok kezelésében

elmarad a LibreOffice-tól, bár rendelkezik egy-két olyan fontos tulajdonsággal (pl. táblázatstílusok, függőleges tömbösítés), amivel a LibreOffice még nem.

A TEX szövegszedő rendszer mai változatai számos olyan lehetőséget nyújtanak, amit csak a kereskedelmi kiadványszerkesztők, például bekezdés szintű tömbösítés, optikai margó, OpenType betűtulajdonságok kezelése, sőt sok olyat is, amit azok sem, különösen a matematikai szövegek szedésében. Jellegénél fogva (alapesetben nincs grafikus felhasználói felülete, hanem parancsállományokkal dolgozik) még távolabb áll a klasszikus DTP rendszerektől, mint a LibreOffice, ennek ellenére (vagy éppen ezért) a szedési munka jól automatizálható vele (még ha ezt a összehasonlító táblázat nem is tükrözi). Több hiányossága van a LibreOffice-hoz képest, például a magyar elválasztás, egalizálás, alapértelmezett tipográfia, dokumentumstandard stb. esetében.

A táblázatban a gondolatjel alapesetben hiányzó, vagy csak körülményesen elérhető képességet jelöl. Például az OpenType kiskapitálisokat vagy ugráló számokat egyesével is beilleszthetjük a Scribusban, de ez nem hasonlítható össze azzal, hogy mindez a LibreOffice-ban valódi betűtulajdonságként is beállítható (közvetlenül, vagy karakter- és bekezdésstílusokban), ráadásul alapból rendelkezésre állnak az ilyen betűvariánsokat tartalmazó betűkészletek (Linux Libertine és Biolinum G).

Képesség	LibreOffice	Scribus	InDesign	MS Office	TEX
Elsősorban DTP-re	–	igen	igen	–	–
GUI[2]	igen	igen	igen	igen	–
szabad szoftver	igen	igen	–	–	igen
ingyenes	igen	igen	–	–	igen
fejlesztő[3]	TDF	közösség	Adobe	Microsoft	D. Knuth
céges támogatás	igen	–	igen	igen	–
PDF-kimenet[4]	igen	igen	igen	igen	igen
rétegek	–	igen	igen	–	–
elválasztás[5]	igen	hiányos	hiányos	hiányos	hiányos
fattyúsorok	igen	–	igen	igen	igen
árvasorok címnél[6]	igen	–	igen	igen	–
soregyen	igen	igen	igen	hiányos	igen
a/az mezőkhöz[7]	igen	–	–	–	igen
lexikonok élőfeje	–	–	igen	igen	igen
tömbösítés szintje	sor	sor	bekezdés	sor	bekezdés
függőleges tömbösítés	–	–	igen	igen	igen
optikai margó	hiányos	igen	igen	–	igen
mikrotipográfia[8]	–	–	igen	–	igen
helyesírás	igen	–	igen	igen	–

[2]Grafikus felhasználói felület. A TEX-hez is elérhetőek GUI, sőt WYSIWYG felületek, például ilyen a BaKoMa TEX, LyX és a TEXmacs, de ezekben a kiadványszerkesztőkére hasonlító közvetlen rajzolási lehetőség minimális, amit persze ellensúlyoz a TEX-re jellemző kiemelkedő nyomdai minőséget biztosító képletszerkesztés.

[3]A The Document Foundation bejegyzés alatt álló német alapítvány, amelynek tevékenységét vezető informatikai cégek (Novell, Red Hat Linux, Canonical, Google), egyéb alapítványok (FSF.hu Alapítvány), illetve kormányok (Brazília) támogatják a brazil, francia, holland, német nyelvi közösségekkel összefogva. A matematikai-tudományos lapok szedésében élen járó TEX szerzője, D. E. Knuth lezárta a fejlesztést, már csak hibajavításokat végez, de a különböző változatok, mint a LuaTEX (pdfTEX) és XeTEX aktív fejlesztés alatt áll.

[4]Kiegészítő programokkal a TEX és a korábbi Microsoft Office változatokkal – részben PostScripten keresztül – előállítható PDF, de így ezek nem tartalmaznak olyan LibreOffice által is támogatott extra PDF-lehetőségeket, mint például a könyvjelzők, hiperhivatkozások, űrlapelemek.

[5]Magyar elválasztás. A Scribus és a TEX hasonló elválasztási mintákat tartalmaz, de az első egyáltalán nem támogatja a kettőzött többjegyű mássalhangzók (ggy→gylgy) elválasztását, a TEX pedig csak manuálisan. Az InDesign és a MS Office elválasztóprogramja nem választja el a többszörösen összetett szavak nagy részét, valamint a helyesírási szótárából hiányzó egyéb szavakat.

[6]Beállítható, hogy címsorokat követő bekezdéseknél az árvasorok száma automatikusan nagyobb legyen.

[7]A magyar névelők automatikus kiválasztása és frissítése numerikus mezőhivatkozások előtt.

[8]A betűszélesség szemmel nem érzékelhető változtatásával.

Képesség	LibreOffice	Scribus	InDesign	MS Office	TEX
tezaurusz	igen	–	igen	igen	–
mondatellenőrzés	igen	–	–	igen	–
EPS képek	igen	igen	igen	igen	igen
SVG képek	hiányos	hiányos	–	–	–
tartalomjegyzék	igen	–	igen	igen	igen
karakterstílusok	igen	–	igen	igen	–
bekezdésstílusok	igen	igen	igen	igen	–
táblázatstílusok	–	–	igen	igen	–
keretstílusok	igen	–	igen	–	–
GREP stílusok	–	–	igen	–	–
mesteroldalak	igen	igen	igen	–	–
alapstílusok	igen	–	igen	igen	igen
lábjegyzetek	igen	–	igen	igen	igen
csillagos lábjegyzet[9]	igen	–	–	–	igen
számneves mezők[10]	igen	–	–	–	igen
képletszerkesztés	igen	–	igen	igen	igen
betűtechnológia[11]	Graphite	–	OpenType	hiányos	Metafont
egalizálás[12]	igen	hiányos	hiányos	hiányos	hiányos
∼ írásjelek előtt[13]	igen	–	–	–	–
kurzív korrekció	igen	–	igen	–	igen
optikai alávágás	–	–	igen	–	–
ligatúrák	igen	hiányos	hiányos	hiányos	hiányos
ugráló számok	igen	–	igen	hiányos	igen
igazi kiskapitális	igen	–	igen	–	igen
több betűfokozat[14]	igen	–	igen	–	igen
álló kurzív zárójel	igen	–	–	–	–
ezrestagolás	igen	–	–	–	–
beépített nyelv	Basic, Python	Python	saját	VBA	TEX makró
fájlformátum[15]	ODF	saját	saját	OOXML	saját

9. Példa az EPS-támogatásra

Az EPS (Encapsulated PostScript) a lapleíró PostScript nyelvet használó képformátum. Vektoros és bittérképes ábrákat, vektoros betűkészleteket tartalmazhatnak. A kiadványszerkesztők az EPS képek előnézetét mutatják szerkesztés közben, majd PostScript nyomtatásnál az eredeti EPS képet illesztik be a kimeneti PostScriptbe. Feladat: Készítsünk EPS képet egy PDF kiadvány valamelyik oldalából,

[9] A magyar tipográfiára jellemző *, **, *** csillagozás.

[10] A számokat tartalmazó mezők magyar számnévvé alakítása (a LibreOffice esetében a Graphite betűkészleteinek betűtulajdonságaként).

[11] Alapértelmezett és opcionális betűtulajdonságok kezelése. Az MS Office 2010 csak pár opcionális OpenType betűtulajdonságot kezel, pl. ligatúrákat és ugráló számokat igen, valódi kiskapitálisokat már nem. Az újabb TEX rendszerek az OpenType betűkészletekkel, azok extra lehetőségeivel is megbirkóznak.

[12] A magyar ékezetes betűket alapból egalizálja (Linux Libertine G és Biolinum G).

[13] A magyar nyomdai hagyományok szerint a kettőspont, pontosvessző, kérdő és felkiáltójel előtt nagyobb térközt hagyunk, mint az angolszász tipográfia.

[14] A LibreOffice következő változata a Linux Libertine Display G betűvel, és a magyar és egyéb ékezetes betűket is támogató felső index Graphite betűtulajdonsággal három valódi betűfokozatot tartalmaz a Linux Libertine betűtípus normál betűváltozatából. Az Adobe szintén három, de külön megvásárolható betűfokozatokat biztosít az InDesignhoz. A TEX a MetaFont rendszernek köszönhetően minden betűfokozat minden betűváltozatával rendelkezik.

[15] A LibreOffice alapértelmezett formátuma a legújabb, még nem ISO, de OASIS-nak már elfogadásra benyújtott OpenDocument 1.2., de kiválasztható az ISO OpenDocument 1.1 is. Az MS Office az elhangzott vállalások ellenére nem hogy alapértelmezett formátumként, de választható formátumként sem támogatja mind a mai napig az ISO Office Open XML Strict változatát, hanem csak az elavult átmeneti (transitional) szabványt.

majd ezt illesszük be egy ODF (OpenDocument), illetve PDF kiadványba képként, megőrizve az EPS eredeti minőségét (vektoros betűkészletek stb.).

1. Alakítsuk a PDF kiadvány kívánt oldalát EPS állományra. Nyomtassuk PostScript fájlba a PDF megjelenítő programmal a PDF kiadvány megfelelő oldalát, majd a végeredményt alakítsuk EPS-re a GhostScript program *ps2epsi* parancsával:

   ```
   ps2epsi kimenet.ps kimenet.eps
   ```

2. Az EPS állományt közvetlenül beilleszthetjük a LibreOffice-ban szerkesztett kiadványunkba. Az OpenDocument állomány az eredeti EPS állományokat tartalmazza. Nyomtatásnál válasszuk a PostScript fájlba nyomtatást, hogy az EPS a kimeneti állományba kerüljön.

A ps2epsi nem túl nagy felbontású és fekete-fehér előnézetet rak az EPS állományba. A szintén ingyenes *epstool* csomaggal tetszőleges felbontású és színes előnézeteket is illeszthetünk az EPS állományba, így akár a nem PostScript kimenet esetén is nyomdai minőségben (igaz, bittérképes képként) kerülhet az EPS képek tartalma nyomtatásra, illetve PDF állományba. (PDF-et másképp is kaphatunk, a PostScript kimenet ps2pdf paranccsal történő átalakításával, de így a LibreOffice által ismert PDF-lehetőségek, például hiperhivatkozások hiányoznak, szemben a közvetlen PDF-exportálással.) További egyszerűsítés lehet, ha az alternatív *ps2eps* parancsot alkalmazzuk a ps2epsi helyett, mivel az ezzel kapott előnézet nélküli EPS képhez a LibreOffice készíti el az előnézetet (pl. a GhostScriptet használva), színesben és lényegesen jobb felbontásban, mint a ps2epsi parancs.

10. Példa az OpenDocument használatára

A LibreOffice egyik előnye az OpenDocument formátum, amely valódi segítséget nyújt az olyan összetett kiadványok elkészítésénél is, mint a korábban említett Biblia. Sőt, ilyen méretű és jellegű kiadványok szedésére már nem is nagyon javasolható más módszer, mint az automatikus szövegszedés. Az eredetileg HTML formátumban lévő Károlyi-biblia (Magyar Elektronikus Könyvtár) az ODFpy (OpenDocument Python programkönyvtár) alkalmazással lett OpenDocument formátumúra alakítva. Az OpenDocument állományt pedig a LibreOffice szedte ki, majd került elmentésre PDF-formátumban, a PDF-nézegetőkben is működő kereszthivatkozásokkal.

10.1. ODFpy

Az ODFpy az OpenDocument állományok előállítására és módosítására szogáló Python könyvtár. Eredeti fejlesztője Michael Howitz, a rendszeresen frissített programkönyvtár karbantartója Søren Roug, az EU Európai Környezetvédelmi Ügynökségének munkatársa. A programkönyvtár tulajdonképpen egy burok az OASIS/ISO OpenDocument szabvány körül: lehetővé teszi, hogy pár utasítással helyes ODF állományokat állítsunk elő, illetve módosítsunk, miközben nem kell tartanunk attól, hogy hibás elemeket vagy attribútumokat helyezünk el az XML-ben. Használatához érdemes az OpenDocument szabványt is kéznél tartani (ami egyben maga is jó példa az OpenDocument használatára), l. http://www.oasis-open.org/standards.

10.2. Példa ODF állomány létrehozására

A következő kis Python program egy „Szia, Világ!” sort tartalmazó OpenDocument szöveges (ODT) állományt hoz létre *helloworld* néven:

```
# -*- Encoding: UTF-8 -*-
from odf.opendocument import OpenDocumentText
from odf.text import P

textdoc = OpenDocumentText()
p = P(text = u"Szia, Világ!")
textdoc.text.addElement(p)
textdoc.save("helloworld", True)
```

10.3. Betűkészletek beállítása

A betűkészleteket és az opcionálisan használt Graphite betűtulajdonságokat előre deklarálnunk kell:

```
from odt.style import FontFace
def set_font(fname):
    textdoc.fontfacedecls.addElement((FontFace(name=fname,
fontfamily=fname, fontfamilygeneric="roman",fontpitch="variable")))
set_font("Linux Libertine G")
set_font("Linux Libertine G:sups=1&pnum=1")
```

10.4. Címsor beállítása

Illesszünk be egy Címsor 1 stílusú címsort, amit a megfelelő bekezdés- (középre igazítás) és karakterformázással (a korábban beállított betűk valamelyike félkövér betűváltozatban, piros színben, 16 pontos betűméretben) láttunk el:

```
from odf.style import Style, FontFace, TextProperties, ParagraphProperties
h1 = Style(name="Heading 1", family="paragraph")
h1.addElement(ParagraphProperties(textalign="center"))
h1.addElement(TextProperties(fontsize="16pt",
fontweight="bold", color="#FF0000",  fontname="Linux Libertine G"))
textdoc.styles.addElement(h1)
textdoc.text.addElement(H(outlinelevel=1, text=u"Főcím", stylename=h1))
```

10.5. Szövegkeretek beillesztése

A következőben a margóra helyezünk egy olyan szövegkeretet, ami egy szintben van a horgonypont bekezdésen belüli karakterpozíciójával. A beállításokhoz egy új keretstílust hozunk létre „F” néven. A szövegkeret tartalma egyszerűen „szöveg”.

A keretelemen belül egy szövegdoboz elem található. Ezek méretét minimálisra vettük, hogy szélességük a szövegtartalomnak megfelelően változzon. Ezen a módon elérhető, hogy tükrözött oldalak esetén is a margón elhelyezett szöveg fix távolságra legyen a bekezdések szövegétől (a páros-páratlan oldalaktól függő balra, vagy jobbra igazítás csak program útján állítható be, az ODF-ben nem, ezért ezt a keret változó méretével helyettesítjük, ami egysornyi margóra helyezett szöveg esetén ugyanazt eredményezi). A LibreOffice hatékonyságát jól jelzi, hogy képes a Biblia 30 ezer kereszthivatkozását tartalmazó 13 ezer szövegkeretet megjeleníteni, és az eredményt PDF-formátumban elmenteni. (Érdekességképpen, az InDesign felhasználói kézikönyve figyelmeztet arra, hogy ne használjunk sok szövegkeretet.)

```
from odf.style import GraphicProperties
from odf.draw import Frame, TextBox
Fstyle = Style(name="F", family="graphic", parentstylename="Standard")
```

```
Fstyle.addElement(GraphicProperties(horizontalrel="page-end-margin",
horizontalpos="from-inside", x="0.3cm",
verticalrel="char", verticalpos="top"))
textdoc.styles.addElement(Fstyle)
p = P()
f = Frame(name="keret", stylename=Fstyle, anchortype="char",
x="0.3cm", zindex="0")
p.addElement(f)
tb = TextBox(cornerradius="1mm", minwidth="0.2in", minheight="0.3cm")
f.addElement(tb)
tbp = P(stylename="Margin", text=u"szöveg")
tb.addElement(tbp)
```

10.6. További példák

Dokumentált példa a Biblia kiszedésére: http://www.numbertext.org/biblia
ODFpy honlap: http://odfpy.forge.osor.eu/

11. Hogyan tovább?

Ahogy az összefoglaló táblázatból kiderül, a LibreOffice már most számos egyedülálló DTP képességgel rendelkezik, és nemcsak a magyar tipográfia vonatkozásában. A párizsi LibreOffice konferencián nemrégiben elhangzott előadás[16] összefoglalta a fő fejlesztési feladatokat, és sikerült a szakma[17] és a LibreOffice fejlesztők figyelmét is felkelteni, felmerült például a Microsoft Publisher importszűrő elkészítésének gondolata. Viszonylag egyszerűen kivitelezhető megoldási javaslat készült az OpenType betűtulajdonságok és a rétegek kezelésére, vagy ami a LibreOffice-ra nézve szinte kötelező fejlesztésnek számít, az OpenDocument szabvány számos olyan, a LibreOffice által még nem, vagy kevésbé támogatott funkciót tartalmaz (pl. táblázatstílusok, lekerekített szegélyű szövegdobozok), amelyek várható megvalósításával a LibreOffice továbbhaladhat a megkezdett úton, nemcsak egy szabad és nagy tudású, hanem versenyképes kiadványszerkesztőt nyújtva mindenkinek.

12. Hivatkozások

Jegyzet: http://www.numbertext.org/libreoffice
LibreOffice Graphite betűkészletek: http://www.numbertext.org/linux/index_hu.html
LibreOffice híroldal: http://www.libreoffice.hu

[16] L. http://libreoffice.hu/2011/10/17/towards-desktop-publishing-%E2%80%93-angolul-es-magyarul/
[17] http://libregraphicsworld.org/blog/entry/libreoffice-is-diving-into-desktop-publishing

Zorp, a pokoli operátor tűzfala

Pfeiffer Szilárd

Kivonat

Az előadás célja a Zorp protokollelemezési és -módosítási képességében rejlő számos lehetőség közül néhány olyan felvillantása, amelyek a mindennapi rendszergazdai gyakorlatban előforduló problémákra adnak megoldásokat.

Ilyenek lehetnek a vírusszűrés különböző (HTTP, SMTP) protokollokban, protokollelemek szűrése, módosítása, tiltása (pl: referer host, user-agent headerek értékének cseréje HTTP forgalom, csak read-only parancsok engedélyezése FTP protokoll esetén), SSL titkosítás kibontása, szerver audit lehetőségek.

Tartalomjegyzék

1. Mi az a Zorp? 124
- 1.1. Protokollelemzés . . . 124
- 1.2. Proxy tűzfal . . . 124
- 1.3. Moduláris szerkezet . . . 124
- 1.4. Nyílt forrás . . . 125

2. Mire jó a Zorp? 125
- 2.1. Hozzáférés-vezérlés . . . 125
- 2.2. Adatszivárgás megelőzése . . . 125
- 2.3. Interoperabilitás . . . 126
- 2.4. Tartalomszűrés . . . 126
- 2.5. Audit . . . 127
- 2.6. Flexibilitás . . . 127

3. Hogyan működik a Zorp? 127

1. Mi az a Zorp?

Tömören fogalmazva a *Zorp* egy nyílt forrású mély protokollelemző proxy tűzfal, ami roppant tudományosan hangzik, vagyis akár a pokoli operátor kifogásnaptárában is szerepelhetne, de a benne rejlő lehetőségeket látva inkább a feltétlenül használandó szoftverek listájára kerülne fel. Lássuk miért! Először is az imént említett mágikus kifejezés kulcsszavait sorra véve próbáljuk meg bizonyítani miért!

1.1. Protokollelemzés

Funkciójukból fakadóan a tűzfalalkalmazások mindegyike képes a hálózati forgalom bizonyos mérvű elemzésére, lévén ennek hiányában nem tudnánk a forgalom szabályozására vonatkozó feltételeket megadni. Ez a *Zorp* esetén sincs másképp. A különbség az egyes szoftverek között az elemzés mélységében mutatkozik meg. Míg például a *Netfilter* esetén az oly sokat emlegetett *ISO*/*OSI* modell negyedik – azaz szállítási – rétegén túli elemeket nemigen használhatjuk fel a hálózati forgalom szabályozásánál, addig a *Zorp* esetén akár a legfelső – azaz az alkalmazási – réteget górcső alá véve is hozhatunk döntéseket. A döntés vonatkozhat a szolgáltatás egészére, vagyis tilthatjuk, vagy engedélyezhetjük például egy *FTP* szerver teljes elérését a felhasználók egy csoportja számára, de lehet szó arról is, hogy letölteni engedünk, feltölteni viszont már nem, és az előbbit is csak akkor, hogy ha a letöltendők vírusmentesnek bizonyultak.

1.2. Proxy tűzfal

Csaknem minden, ami ezen kifejezésről eszünkbe juthat, igaz a *Zorp* kapcsán is. Elsőként mindenképpen a *proxy* szerverek azon sajátossága, hogy beékelődnek a kliens és a szerver közé, elszeparálva ezáltal a hálózati kommunikáció két szereplőjét egymástól, és minden egyes kérést, illetve választ újrafogalmaznak a két szereplő között. A *Zorp* ebben a tekintetben annyival több versenytársainál, hogy mindezt alkalmazásszinten tudja megtenni, a felhasználás különböző módjai (*forward proxy*, *reverse proxy*) mellett. Ehhez szükségesek a *C* nyelven implementált, viszont *Python* nyelven konfigurálható, bővíthető protokollelemző osztályok (a *Zorp* terminológiájában *proxy*), melyekből a nyílt forrású változat esetén kilenc, míg a kereskedelmi verzióban huszonöt áll rendelkezésre.

1.3. Moduláris szerkezet

A *Zorp* kétségkívül legnagyobb előnye a testreszabhatóság, amely a szoftver szerkezetének moduláris felépítése nélkül nem volna lehetséges. A mindennapi használat során ez annyit jelent, hogy ha nincs szükségünk egy adott proxy kapcsán speciális működésre, akkor gyakorlatilag szinte semmit nem kell tennünk azért, hogy élvezhessük a hálózati protokoll applikációs szintig történő elemzését. Ha viszont ennél többre van igényünk, kontrollálni kívánjuk az adott forgalmat, akkor is csak arra kell összpontosítanunk, amit konkrétan elérni szeretnénk (pl: egy *HTTP* header módosítása), minden egyebet készen kapunk. Ezeken túl viszont arra is van lehetőség, hogy a *Zorp* csak a hálózati kapcsolatot kezelje, és a protokoll teljes elemzését magunk végezzük függetlenül a gyárilag meglévő *proxy*któl.

A szállítási rétegben megvalósított biztonsági protokollok (*SSL*/*TLS*) implementációja egy önálló alrendszert, ha úgy tetszik modult képez, így az ezekhez kapcsolódó beállítások a konkrét applikációs protokolltól függetlenül tehetőek meg, ami lehetőséget biztosít arra is, hogy egy általunk implementált protokollértelmező *SSL* kapcsolaton belül és kívül egyaránt működhessen. A *Zorp* alkalmas külső eszközökhöz, modulokhoz való integrációra is, melynek lévén viszonylag könnyen valósíthatóak meg spam-, illetve vírusszűrő megoldások.

1.4. Nyílt forrás

Lévén egy szabad szoftveres konferencia előadásának kiegészítő anyagáról van szó, nagy meglepetést nem okozhat, hogy a *Zorp* nyílt forrású termék, ugyanakkor néhány kérdést mégis fontos tisztázni, ha más nem, a licenchuszárok kedvéért. Egyrészről, hogy a szerzői jogi védelem pontosan milyen formájáról van szó, másrészről, hogy kizárólagos-e ez a licencelés. Az első kérdésre kettős válasz adható, lévén maga a *Zorp* is két összetevőre oszlik, magára a tűzfal applikációra, illetve egy hozzá kapcsolódó függvénykönyvtárra. Mindkettő az FSF által kiadott licenc – az előbbi (`Zorp`) *GPL*, míg az utóbbi (`libzorpll`) *LGPL* – hatálya alatt érhető el. A második kérdésre adott válasz egy kifejezésben foglalható össze; *dual-license*, vagyis létezik egy nyílt forrású (*Zorp*/*Zorp GPL*), illetve egy kereskedelmi (*Zorp Professional*) változat, a megfelelő licencelési, illetve számos technikai különbséggel, melyekről később még esik szó.

2. Mire jó a Zorp?

Ha egy marketing anyag sorai közé tévedt volna a nyájas olvasó, a fent megfogalmazott költő kérdésre a válasz bizonnyal az lenne; mindenre. Így viszont némi szerénységet felmutatva szűkítjük ezt a halmazt belátva, hogy a *Zorp* közel sem mindennek a teteje, viszont minden funkcióban segítségünkre lehet, ami a mély protokollelemző proxy tűzfal varázsos meghatározás alapján elvárható. Lássuk mik lennének ezek.

2.1. Hozzáférés-vezérlés

A proxyszerverek ezen alapvető funkciója esetén a *Zorp* annyiban különleges, hogy itt is el tud szakadni az alacsonyabb *ISO*/*OSI* rétegektől. Míg más eszközöknél csak a hálózati réteg attribútumai – *IP* címek, illetve alhálózatok – segítségével szabályozhatjuk a szolgáltatásokhoz való hozzáférést, addig a *Zorp* egy saját rendező mechanizmusa alapján. Ez a *zóna*, ami alhálózatok szabadon csoportosítható halmazát jelenti, melyek fába szervezhetőek. A *zóna*struktúra szintje között a szolgáltatások elérésének szabályai öröklődnek, de kivételek is megadhatóak, így egy *IP* alhálózatoktól független adminisztratív hierarchia hozható lére, ami nem a hálózati eszközeink elrendezését, hanem a hozzáférés szab@?lyait tükrözi.

A *kik* mellett a hozzáférés-vezérlési rendszer megalkotásakor a *mit* és a *hogyan* kérdéseire is választ kell adnunk, vagyis nem csak az határozandó meg, hogy kik számára érhetőek el az erőforrások, hanem azt is, hogy milyen feltételek mellett. Lehet szó egész egyszerűen csak arról, hogy egy szerver elérése esetén minden egyes végrehajtott műveletről bejegyzés készül a rendszernaplóba. Vagy tilthatjuk például a szerver, illetve a kliens olyan funkcióit, melyek inkompatibilitást okoznak. Szűrhetjük a protokoll némely elemeit, módosíthatjuk azok értékeit, hogy elkerüljük érzékeny adatok kiszivárgását. Ellenőrizhetjük az adattartalom vírusmentes mivoltát. Kibonthatjuk, majd visszacsomagolhatjuk a forgalmat védő titkosítást. Ebből adnak ízelítőt a következő fejezetek.

2.2. Adatszivárgás megelőzése

Számos protokoll esetén léteznek olyan – egyébiránt teljesen szabályos, következésképp a tűzfalak által alapvetően nem szűrt vagy tiltott – protokollelemek, melyek a kliensen futó szoftverekről, esetleg annak hálózati beállításairól szivárogtatnak ki adott körülmények között érzékeny információkat. Erre példa a `user-agent` mező – egyebek mellett – a *HTTP* protokoll fejlécében, ahol is ennek értéke a használt böngésző típusa, illetve verziója, mely akaratunkon kívül jut el az általunk éppen meglátogatott webszerverhez.

Ehhez közel azonos módon szolgáltathatja ki a böngésző a proxybeállításokat, gépünk *IP* címét, vagy épp az előzőleg látogatott oldal (amiről az adott szerverre jutottunk) *URL*-jét. Ehhez hasonló megoldások azonban nem csak a *HTTP* protokollban szerepelnek, hanem másutt is, melyekről érdemes tudni és adott esetben érdemes őket tiltani. Erre a *Zorp* könnyen használható és rugalmas megoldást ad.

2.3. Interoperabilitás

A fenti példánál maradva a böngésző típusának nem csak az elrejtése lehetséges, de a „meghamisítása" is, ami az interoperabilitás megteremtéséhez annyiban segít minket hozzá, amennyiben találkozunk azzal az – egyébiránt egyre ritkábban előforduló – esettel, amikor is egy webszerver annak ellenére is kikötést tesz a böngésző típusára, hogy erre érdemben alátámasztható oka nincs. Az ilyen helyzet könnyedén feloldható azáltal, hogy a böngésző által küldött `user-agent` mező értékét úgy változtatjuk meg, hogy az a webkiszolgáló számára elfogadható legyen.

Éppúgy kellemetlenségeket okozhat bizonyos – általában nem mai keletű – szoftvereknél a titkosított forgalom kezelésének hiánya, főként ha egy nem megbízható hálózaton keresztük kell átjuttatnunk az adatokat. Erre a helyzetre természetesen számos megoldás létezik, ugyanakkor ha a problémát megfejeljük azzal, hogy a forgalmat egy tűzfalon is szeretnénk keresztül vezetni, akkor válik igazán hasznossá a *Zorp* azon szolgáltatása, mely alkalmassá teszi a kliens és a szerver irányába más titkosítási metódus (*TLS*, *SSL*) használatára. Ennek szélsőséges este, amikor a *Zorp* csak az egyik – megbízhatatlannak tartott – irányba végez titkosítást, míg a másik – megbízhatónak vélt – irányba nem.

Fentiekhez nem feltétlenül elegendő csupán a titkosított csatornák kiépítésének, valamint újraépítésének képessége, lévén a protokoll részeként is kezdeményezhető titkosított kapcsolat kiépítése, ami az interoperabilitás egy újabb területére, a funkciók elrejtésére vezet át minket. Amennyiben azt szeretnénk, hogy bizonyos – mind a kliens mind a szerver által ismert – funkciók ne legyenek elérhetőek – mondjuk valamilyen inkompatibilitási probléma miatt –, ezt is módunkban áll a *Zorp* segítségével megtenni.

A titkosítási példát tovább gördítve adott a lehetőség, hogy az *SMTP* protokoll *feature* listájából, kiemeljük a *STARTTLS* elemet, ami megakadályozza, hogy a kliens és a szerver ilyen módon kommunikáljanak egymással. Az *SSL* beállítások egyes kombinációinál – például ha a szerver irányába kötelező az *SSL* – azt a *Zorp* automatikusan megteszi.

2.4. Tartalomszűrés

Nincsen tűzfal tartalomszűrés nélkül. A szabály alól a *Zorp* sem kivétel, még akkor sem, ha a *Zorp* önmagában csak korlátozottan alkalmas ezen feladat ellátására. A hangsúly az önmagában kifejezésen van, hiszen kiegészítve egyéb eszközökkel, mint vírus-, spam-, *URL*-szűrő, a *Zorp*, mint suszter, maradhat a kaptafánál, vagyis nem tesz egyebet, mint értelmezi az adott protokollt, ennek segítségével kiemeli a hálózati forgalomból azt a részt (*URL*, letöltendő fájl, levél, ...), melyet továbbadva a alkalmas szoftvernek, annak visszajelzése alapján döntést hozhat, hogy visszautasítja, átengedi, karanténba helyezi, vagy csak naplózza az eseményt függően akár más beállításoktól, vagy a forgalom egyéb körülményeitől. Ehhez nem kell egyebet tennünk, mint egy illesztőprogramot helyezni tűzfalunk és az általunk választott tartalomszűrést biztosító eszköz közé, mely tudatja előbbivel az utóbbi által megállapítottak alapján hozott döntést.

2.5. Audit

Egy hálózat adminisztrációja során az erőforrások elérését szabályozó rendszer felállítása csupán az első lépés, ennek a rendszernek a felügyelete adja a munka nagyobb részét, melyből a szabályok fejlesztése, átalakítása következik majd. Mindenekelőtt azt kell megtudnunk, mi az ami a hálózatunkban aktuálisan zajlik. Egyrészről mik azok az események, amik a megalkotott szabályrendszer áthágására irányultak. Másrészről az engedélyezett akciók közül melyek következtek be és milyen formában. A *Zorp* mindkét típushoz tartozó eseményekről bejegyzéseket ír a rendszernaplóba.

Az utóbbi eset az, ahol az applikációs szintű protokollelemzésnek hasznát láthatjuk, hiszen az egyes protokollok parancsainak szintjén van módunk rálátást szerezni a hálózatban zajló eseményekre, illetve rögzíteni azokat a rendszernaplóba. Ennek mind belső, mind külső audit során komoly hasznát vehetjük, hiszen – a naplózás megfelelő beállításai mellett – igazolható, hogy mely események történtek meg és melyek nem egy adott időszakban. Emellett természetesen forgalmi-, illetve eseménystatisztikák alapjául szolgálhatnak ezek a naplóbejegyzések.

2.6. Flexibilitás

Az eddig leírtak kész, vagy legalábbis félkész megoldásként a *Zorp* részét képezik, ugyanakkor a *Zorp* erejét épp az adja, hogy szabadon és rugalmasan bővíthető saját céljaink szerint. Ehhez a *Zorp* azon sajátosságát használhatjuk ki, hogy a már meglévő eszközöket (*proxy*k) könnyen újrahasznosíthatjuk, vagyis az egyes protokollelemzőket nem kell újra megírnunk, elegendő azok *Python* nyelvű változatait újra felhasználva csak a szükséges kiegészítéseket megtenni. Még abban az esetben is igaz ez, ha teljes egészében magunk kívánjuk a hálózati forgalmat feldolgozni – amit egyébiránt megtehetünk úgy is, hogy hozzá megírjuk a alkalmas *C* nyelvű *proxy*t –, mivel létezik egy erre a célra szolgáló *Python* osztály (*AnyPy*). Minden, ami az *OSI* modell alsóbb rétegeiben zajlik, implementált a *proxy*ban, nekünk csak az ezen felüli – leginkább az alkalmazási – réteg részleteire kell koncentrálnunk.

Ami elmondható, hogy a már meglévő eszközökből való öröklés és a nyílt forrás adta előnyök révén számos igény kielégíthető.

3. Hogyan működik a Zorp?

Ennek az anyagnak a keretei nem teszik lehetővé az egyes konfigurációk részletes ismertetését. A működés részleteinek tekintetében érdemes a *Zorp* GitHub[1] oldalát meglátogatni, ahol számos konfigurációs példa[2] érhető el. Ezen felül virtuális gépek[3] révén egy teljes működő rendszerhez juthatunk, ahol a valós működés tesztelhető.

[1] http://github.com/balabit/zorp
[2] https://github.com/balabit/zorp-examples
[3] http://people.balabit.hu/szilard/zorp-gpl/virtual-machines/

Nyílt forráskódú megoldások a közigazgatásban

Szegfű László

Kivonat

A magyar költségvetés számára rendkívül nagy teher a közszolgálati intézmények éves szoftverlicenc ellátása, legyen szó a közigazgatás vagy akár a közoktatás számára vásárolt termékek áráról. A közigazgatásban az informatikával támogatható feladatok jellemzően nem specializáltak, így lehetővé válik olyan megoldások használata, amelyek olcsóbban vagy akár ingyenesen is beszerezhetők. A közigazgatási informatikában használt programok nyílt forráskódú rendszerekre történő cseréjével jelentős költségmegtakarítás érhető el. Szeged M. J. Város példája mutatja, hogy csak a polgármesteri hivatalban használt szoftverek ilyen cseréjével 8 év alatt több száz millió forint értékű licenc díjat lehet megtakarítani. A cserék természetesen valamennyi lemondással járnak, de a teljes közigazgatás átalakításával a főként kompatibilitási problémák jelentős része kiküszöbölhető.

Tartalomjegyzék

1. Bevezetés **130**
1.1. Közigazgatási szoftverek 130

2. Nyílt forráskódú rendszerek bevezetésének célszerűsége **130**
2.1. Költséghatékonyság 130
2.2. Függetlenség a szállítótól 130
2.3. Interoperabilitás 131
2.4. Integrálás, migrálás 132
2.5. Átlátható kód 132
2.6. Egyenértékűség 132
2.7. Licencek 133
2.8. Vírusvédelem, biztonság 133

3. A nyílt forráskódú rendszerek bevezetését gátló tényezők **133**
3.1. Az újtól, a nyílt forráskódtól való félelem, dolgozói ellenállás 133
3.2. Kompatibilitási problémák 134
3.3. Szigetalkalmazások és központi szoftverek használata 134
3.4. Nem támogatott hardver 135

4. Megoldások **135**

1. Bevezetés

A magyar közigazgatásban elterjedt szoftvertermékek vásárlása, fenntartása rendkívüli terhet ró a mindenkori költségvetésre. Többször felmerült a kérdés, hogyan lehetne költséghatékony megoldásokat találni a kiadások mérséklésére. Ez a kérdés nem csupán olyankor kell, hogy felmerüljön, amikor gazdasági válság van, a gondos gazda szerepét az államnak, a közigazgatási szerveknek és a költségvetési intézményeknek mindenkor felelősséggel el kell látnia. Szeged Megyei Jogú Város Önkormányzata több, mint 8 éves tapasztalata alapján megmutatható, hogy gondos előkészítést követően létrehozható jól működő, olcsó infrastruktúra, elsősorban nyílt forráskódú megoldások alkalmazásával.

1.1. Közigazgatási szoftverek

A közigazgatási szerveknél a legtöbb hivatalban végfelhasználói szempontból jellemzően kevés szoftverre van szükség, így például: operációs rendszerre, irodai alkalmazáscsomagra, irodai háttértámogatást nyújtó rendszerekre (ezek lehetnek kisebb szigetszerű célszoftverek, de lehetnek akár nagyobb adatbázisrendszerek is), elektronikus levelezési rendszerekre és internetböngésző szoftverre. Ezen kívül kevés számban előfordulhatnak speciális célszoftverek, tervezőrendszerek, csoportmunkatámogató szoftverek, grafikus vagy egyéb programok.

Az irodai háttértámogatást nyújtó szigetalkalmazásokat, vagy nagyobb adatbázisrendszereket vizsgálva megállapíthatjuk, hogy ahány hivatal, ahány ügye, annyi megoldás létezik. Ezek a szoftverek többségükben a papír alapú megoldás mellett működnek, ennek legfőbb oka, hogy a munkafolyamat kapcsán keletkező aláírásokat nehezen lehet hiteles formában – mindenki megelégedésére – elektronikusan kiváltani. Ezek a programok általában rendkívül drágák, és az adatok kinyerése, feldolgozása, más rendszerekbe történő átemelése rendkívül körülményes.

Az informatika működéséhez szükségszerűen a szerveroldali megoldások is hozzátartoznak. Itt jellemzően ugyanazok a feladatok adódnak, mint minden más esetben: többek között a fájlok tárolása, adatbázisok üzemeltetése, levelezés, internetes portál működtetése, adatbiztonság kialakítása stb.

2. Nyílt forráskódú rendszerek bevezetésének célszerűsége

2.1. Költséghatékonyság

A nyílt forráskódú rendszerek bevezetése mellett és ellen egyaránt számos érv sorolható fel. A mellette szólók közül a legfőbb érv minden esetben a költséghatékonyság. Általában mindenki olyan termékeket vásárol, ami az ő igényeinek megfelelően a legjobb ár/érték arányú. Akinek több pénze van, jobb, kényelmesebb, többet tudó terméket tud vásárolni, akinek kevesebb forrása van, értelemszerűen neki megfelelőt keres. Nem szabad elfelejteni azonban, hogy a közigazgatás az adófizetők pénzéből gazdálkodik, így elvárható, hogy mindenkor az igényeinek megfelelő legolcsóbb – adott esetben ingyenes – terméket és így szoftvert szerezze be. Tehát amennyiben létezik ingyenes alternatíva, a közigazgatási szereplők a gondos gazda szerepével élve kötelesek azt beszerezni.

2.2. Függetlenség a szállítótól

A magas költségek nem csak az általános szoftverek árában követhetők nyomon, az irodai folyamatok informatikai támogatására készített egyedi programok ára és fenntartása is rendkívül drága. A költségeket általában az határozza meg, hogy a piacon régóta jelen lévő beszállítók a kezdeti időkben – néhány kivételtől eltekintve – indokolatlanul magas fejlesztési és licencdíjakat állapítottak

meg. Ezt megelőzően általában ingyenesen felajánlották a szoftvereket használatra, majd amikor a felhasználók megszokták a felületet, és kellő mennyiségű adatot vittek be a rendszerbe, a fejlesztők bejelentették a további felhasználás feltételeit, mintegy kész tények elé állítva a hivatalokat. A jogszabály-módosítások vagy a felhasználói igények változása okán szükségszerű drága fenntartási költséget fizetni a fejlesztőknek. A már meglévő szoftverek módosításának vagy fejlesztésének a költsége pedig gyakorlatilag annyiba kerül, amennyit a fejlesztő ezért a munkáért elkér. A fejlesztéseket ezután – lévén a vagyoni jogokat nem ruházza át – bármely másik, hasonló munkát végző közigazgatási szervnek is el tudja adni, anélkül, hogy újabb munkaórákat fektetne a fejlesztésbe. Az így elvégzett munka nem csak indokolatlanul magas költséggel jár, de többszörösen kifizetésre kerül az adófizetők pénzéből. Külön kockázatot jelent az időközben jogutódlás nélkül megszűnő, vagy érdektelensége okán a munkát egészen egyszerűen megtagadó vállalkozó.

A fent felsorolt programok általában erősen platformfüggők, így az esetleges olcsóbb, vagy ma már nagyon népszerű, de nem szokványos felületeken nem futtathatók. Ennek az az eredménye, hogy a többletköltségeken felül a technológiai fejlődést is erősen hátráltatják. A "rátermettebb" fejlesztők megpróbálják ugyan átültetni az elavult technológiát valamilyen grafikus felületre, de ezzel sajnos a platformfüggőség nem szűnik meg, csak más felülethez köti a felhasználót.

Köztudott, hogy minden rendszert egyszer ki kell cserélni, mert elavul. Ezért a jövőben olyan rendszer(ek) fejlesztésére kell törekedni, amelyek forráskódja és vagyoni joga "birtokon belül" marad. Ezek a megoldások lehetnek akár publikált nyílt forráskódú, akár az állam (vagy az adott közigazgatási szerv) tulajdonába kerülő forráskódú és vagyoni jogú szoftverek. Ilyen megoldások használata esetén a fejlesztő a ténylegesen elvégzett munkájáért kapja meg a megfelelő díjazást, és módosítási vagy fejlesztési igény esetén az adott közigazgatási szerv eldöntheti, hogy a munkát maga végzi el, vagy az adott fejlesztőt bízza meg, vagy megversenyezteti az árakat más fejlesztőkkel is. Értelemszerűen egy nagyobb munka esetén nehéz olcsóbb árat ajánlani az eredeti fejlesztő áránál, de megakadályozható az indokolatlanul magas, és többször beszedett munkaköltség. A verseny tudata önmagában árletörő hatású. Ha nem sikerül olcsó megoldást találni, a forráskód birtokában a rendszerek akár ki is válthatók.

2.3. Interoperabilitás

A növekvő információigénnyel arányosan egyre nagyobb az igény a meglévő adatbázisrendszerek átjárhatóságára, összekapcsolására a gyorsabb és pontosabb, valamint a lehető legszélesebb körű adatszolgáltatásra, illetve az adatok minél kevesebb munkával és minél kevesebb adatbázisban történő tárolására.

A zárt kódú rendszerek használatának interoperabilitási szempontból több kockázata is felmerül. Két rendszer illesztése mindkét rendszer fejlesztőjének, és a megrendelőnek egyidejű egyetértésén és munkáján alapul. Ezt összehangolni rendkívül nehéz, és ennek a megvalósulása és költsége meglehetősen bizonytalan. Felmerülő probléma esetén a fejlesztők "egymásra mutogatnak".

Nagyobb a probléma azonban, ha nem csak két rendszert kell összekapcsolni. Ha egy központi rendszerhez több, egymástól eltérő szoftvert használó hivatal alkalmazott megoldását kell hozzáilleszteni, ezek a problémák – és ezzel egyidejűleg a hibalehetőségek is – hatványozottan jelentkezhetnek.

A forráskód(ok) birtokában az átjárhatóság megoldása is könnyebb, mivel megismerhetjük a fogadó rendszer működését, ehhez alakíthatjuk a saját rendszerünket. Mindkét oldalon kevesebb együttműködés szükséges a hatékony munkához, a kód birtokában egyszerűbben visszafejthetők a hibák is.

2.4. Integrálás, migrálás

Ahogy korábban szó volt róla, felelősen gondolkodó informatikus tudja, hogy minden rendszert egyszer ki kell váltani. Az adatok kinyerhetőségét ezért minden újonnan fejlesztett rendszernél biztosítani kell. Ehhez szükség van a kiváltandó rendszer fejlesztőjére, hiszen zárt kódú megoldásoknál nem ismerjük az adatmodellt, az adatbázis-szerkezetet és a számolt adatokat, illetve azok összefüggéseit és a többit.

A zárt rendszerek esetében általában a kiváltandó rendszerek fejlesztője egyáltalán nem érdekelt a rendszer kiváltásában, így jellemzően az ilyen munkálatokért olyan költségek kerülnek felszámolásra, amelyek kifizetése elbizonytalanítja vagy ellehetetleníti a szoftver kiváltását. A tapasztalatok szerint többéves fenntartási és licenc költség megtérülését biztosan bele kell számítani az ilyen munkálatok árába. Külön gondot okoz, ha az időközben megszűnt vagy ellehetetlenült vállalkozásokat nem lehet felderíteni vagy rávenni a migrálásban történő segédkezésre.

Adathibák, konverziós hibák, nem konzisztens adatok, és a többi hiba esetén azok javítása is szintén "egymásra mutogatást" eredményezhet a régi és az új rendszer fejlesztője illetve a megrendelő között.

2.5. Átlátható kód

A zárt kódú programok esetében a program tényleges működése nem ismerhető meg. A felhasználó "fekete dobozként" látja a szoftvert, nem tudja, hogy az általa bevitt és kinyert adatokon felül még milyen adatok kerülnek bele, és milyen adatok kerülnek ki a programból.

Sokan félnek a nyílt forráskódú megoldásoktól, mert azt gondolhatják, hogy így megismerhetők és ezáltal manipulálhatók a bevitt, és kinyerhetők a szenzitív adatok. Pedig éppen a zárt rendszereknél nem tudjuk, hogy ez megtörténik-e.

A forráskód birtokában a rendszer működése ellenőrizhető. Nem kell feltétlenül arra gondolni, hogy a közigazgatás bármelyik szereplője kész és képes egy nagyobb program forráskódjának ellenőrzésére, de egy szoftver minőségbiztosítása független szakértőkkel is elvégeztethető. Egy "kész" nyílt forráskódú szoftvert pedig olyan közösség fejleszt, akiknek egészen biztosan feltűnik, ha valaki direkt vagy véletlenül "ártalmas" kódrészletet akar a rendszerbe csempészni. A közösségi fejlesztések esetén a hibák felfedezése is nagyobb valószínűséggel megtörténik, hiszen a fejlesztők nem egyféle elgondolás alapján dolgoznak.

2.6. Egyenértékűség

Az elmúlt néhány évben a közösségek által fejlesztett, általános felhasználású szoftverek rendkívül sokat fejlődtek. A közigazgatásban használt, korábban felsorolt szoftvertermékek nagy többségének általában van egy vagy akár több, ilyen megbízható alternatívája. Ahogy operációs rendszer, irodai alkalmazáscsomag, levelező vagy internetböngésző program, úgy akár komoly fotó- és hangszerkesztő vagy tervező rendszer is létezik, megfelelő minőségben.

Az ilyen nyílt forráskódú programok használata általában ingyenes, míg ezek zárt kódú alternatívái jellemzően nagyon sokba kerülnek. Ez az oka, hogy ezek sokszor legfeljebb egy-két számítógépre kerülnek jogszerűen telepítésre, pedig elképzelhető, hogy az érdemi munkavégzéshez ettől több licencre lenne szükség. Ha minden közigazgatási szereplő minden munkájához az ingyenes szoftvereket használná, biztosítható lenne az adatátjárhatóság. Nem lenne szükség nagyon drága célszoftvereket vásárolni csak azért, hogy egymás adatait olvasni tudják. Sok esetben egyébként a nyílt forráskódú program szabvány formátumot használ, így biztosítható az adatok későbbi megnyitása is, míg egy-egy zárt kódú megoldásnál tapasztalható, hogy már a korábbi verziójú szoftver által elkészített adatok sem olvashatók hibátlanul.

2.7. Licencek

A szoftverlicencek olyan jogok, amelyekért azért kell fizetnünk, hogy a kész, másoknak is eladott szoftverterméket korlátozott vagy jó esetben korlátlan ideig, korlátozott számban, területen használhassuk. Az ilyen fizetési kötelezettség mögött rejlő érdemi teljesítést sokan megkérdőjelezik.

A licencelt szoftverek jellemzően meghatározott hardverre, általában egyszer, egy felhasználó számára telepíthetők, előfordul, hogy még hardverkulcsot is kell használni. Ezeket a programokat azonosító számuk alapján, pontosan, napra készen nyilván kell tartani. Ellenőrzéskor a legkisebb eltérés vagy hiányosság esetén is indokolatlanul nagy retorziókra kell számítani.

Nagyobb rendszerek esetén a rendszergazdák azonos típusú konfigurációk telepítésére klónozási technológiát alkalmaz(ná)nak. Ezzel a megoldással a napokig tartó telepítgetések és beállítgatások pár perc alatt elvégezhetők. Sajnos az ilyen klónozási technológiák azt a veszélyt hordozzák, hogy a licencek sorozatszáma nem változtatható meg, így hiába van a felhasználó birtokában megfelelő számú licenc, a sorozatszámok eltérése miatt azokat nem jogszerűen használja. Az azonos időben beszerzett, azonos konfigurációk általában egyidejűleg hibásodnak meg, így a rendszergazdáknak az adminisztrációra is indokolatlanul nagy figyelmet kell fordítani. A feladatot nem oldják meg, legfeljebb könnyítik azok az alkalmazások, amelyek összegyűjtik a gépek tulajdonságait, hiszen a korábbi leltárral minden esetben egyeztetni kell az eltéréseket.

Ezzel szemben a nyílt forráskódú szoftvereket nem szükséges az egyedi licenc- azonosítószám alapján egyenként nyilvántartani. A szoftverek általában mindenféle korlátozás nélkül telepíthetők, így nem kell aggódni lejárt, áttelepített, más gépen használt programok miatt. Nem kell a használati jogokat megújítani, és nem kell időközönként az adófizetők pénzéből újabb szoftverlicencekre költeni.

A nyílt forráskódú szoftverek többségét nem kell online vagy telefonos ügyfélszolgálaton keresztül regisztrálni ahhoz, hogy működésre bírjuk, nem kell magyarázkodni, ha többször újra szeretnénk telepíteni az adott programot.

2.8. Vírusvédelem, biztonság

Mint az ismeretes, feltörhetetlen rendszer nincs. Jóval kisebb azonban a nyílt forráskódú rendszereket érő támadások száma, legyen az vírus- vagy crackertámadás. A közösségek által fejlesztett rendszerek esetén "több szem többet lát" alapon javítják a hibákat, így a tapasztalatok azt mutatják, hogy jóval kevesebb hiba marad az ilyen rendszerekben, mint a zártakban, amikbe csak a fejlesztők látnak bele. Tapasztalat az is, hogy a nyílt rendszerekkel telepített kliensekkel gyakorlatilag nincs probléma a többi számítógéphez képest.

Az ismert vírusok, férgek, trójaik nagyon nagy százaléka nem fut ezeken a rendszereken, így komoly vírusvédelmi rendszer árát is adott esetben meg lehet takarítani.

3. A nyílt forráskódú rendszerek bevezetését gátló tényezők

3.1. Az újtól, a nyílt forráskódtól való félelem, dolgozói ellenállás

Általában igaz, hogy az emberek tartanak az újtól. A régi, megszokott dolgokat szeretik használni, beleértve a munkahelyükön használt szoftvereket is. A legtöbb esetben a váltástól való félelem alapja, hogy előítéletekkel fordulnak a megszokottól eltérő rendszerek felé, akkor is, ha azt egyáltalán nem ismerik (kivételek ez alól a státuszszimbólumként funkcionáló eszközök, amiken mindegy milyen rendszer fut). A legtöbb dolgozó azt a generációt képviseli, akik még nem számítógépen szocializálódtak, így egy-egy szoftver használatának elsajátítása külön gondot okoz számukra. Ezek a dolgozók különösen nagy aggodalommal hallják, ha egy rendszer "nyílt", mivel azt gondolják, hogy

így bárki belátást nyerhet a program által létrehozott adatokba, illetve információt nyerhet a kollégák egyéni munkájáról. Az ilyen feltételezések híre gyorsan elterjed, a felhasználók pedig együttes ellenállással képesek lehetnek a bevezetés ellehetetlenítésében.

Az újtól való félelem nem csak a felhasználókat, de a legtöbb esetben a rendszergazdákat is érinti. A jól megszokott, könnyen telepíthető és használható szerverszoftverek, az ismert problémákkal küzdő kliens oldali operációs rendszerek az adott környezetben mindenképp kényelmet jelentenek. A közigazgatásban dolgozó régi vágású rendszergazdák közül sokan nem nyitottak az új kihívásokra, félelmeik vannak a jelenleg működő megoldások átültetésére egy újabb technológiai környezetbe. Sokan azzal magyarázzák a nyílt forráskódú rendszerekre történő áttéréstől való félelmüket, hogy nem kapnak megfelelő terméktámogatást a nyílt forráskódú rendszerekhez, illetve a megfelelő minőségű szupport költségét kifizetve ugyanakkora költséget jelent, mint, ha megmaradtak volna az eredeti verzióknál. Hallottam már olyan indokot is, hogy egy esetleges meghibásodáskor nyílt kódú rendszerek használata esetén a felelősséget nincs kire áthárítani.

Sajnos a tapasztalataink azt mutatják, hogy zárt kódú szoftverek használatakor felmerülő kisebb-nagyobb hibákat a helyi üzemeltetők is meg tudják oldani, a megold(hat)atlan problémára pedig a gyártó által nyújtott támogatás sem tud kielégítő válasszal szolgálni. A felelősség áthárítása érdekében pedig célszerű a szoftverhez kapott tájékoztatót elolvasni: magam még sosem találkoztam olyan szoftverrel, amelynek a gyártója felelősséget vállalt volna a termék használatából eredő károkért.

Összességében elmondható tehát, hogy azonos tudású szoftverek esetében általában a zárt és a nyílt kódú rendszerekhez ugyanúgy nem kapunk ingyen érdemi támogatást és felelősségvállalást, így a szoftverek között gyakorlatilag a termékek ára tesz különbséget.

3.2. Kompatibilitási problémák

A számítógépet közigazgatási alkalmazása során a legtöbb esetben egyszerűen az írógép kiváltására használják. A használt irodai alkalmazáscsomag "szokásjog" alapján az egyik legelterjedtebb gyártó drága szoftvere. Ennek a szoftvernek a folyamatos frissítése közigazgatási felhasználás szempontjából semmiféle újdonságot nem tartalmaz, ugyanakkor a mindenkor legfrissebb változat használata szükségszerűen magával viszi a többi felhasználót is, mivel az eltérő verziók alapértelmezett formátuma jellemzően kompatibilitási problémákat okoz. Márpedig aki új szoftvert vásárol, egyrészt nem "bűvészkedik" az alapértelmezett formátum átállításával, másrészt nem is akarja a "rosszabb" formátumot használni, annak ellenére, hogy az új formátum érdemben nem nyújt új funkcionalitást. Sajnos ezekkel az alkalmazásokkal készített dokumentumok legtöbbször a korábbi verziójú programokkal sem csereszabatosak, a nyílt forráskódú alternatíva pedig bármilyen tökéletes, konverziós hibák mindig maradnak a rendszerben.

Megoldást az jelentene, ha minden állam- és közigazgatási szereplő szabványos formátumot használna a dokumentumok formátumának megválasztásakor. Így elkerülhetők lennének a kompatibilitási és a későbbi megnyitásból származó problémák.

3.3. Szigetalkalmazások és központi szoftverek használata

Sok hivatalban az átállást nehezíti a régi, zárt rendszerek okozta függőség. A legtöbb ilyen rendszer – ahogyan korábban már szó esett erről – erősen platformfüggő. Ezekbe az évek során számos nélkülözhetetlen adat került, így az interoperabilitás megoldása illetve az adatmigrálás nehézkes és drága folyamat, legtöbbször csak hosszú párhuzamos üzem vagy a rendszer teljes avulása esetén történő csere jöhet számításba. Amíg a platformfüggő szoftverek meghatározzák az alkalmazott informatikai rendszert, addig sem szerver, sem kliens oldalon nem lehet teljes körű cserében gondolkodni.

Az egyedileg gyártott szigetalkalmazásokon túl a közigazgatásban számos olyan központi szoftver használata kötelező, amelyek még régi, karakteres operációs rendszeren futnak. Ezeknek a prog-

ramoknak a futtatását meg lehet ugyan oldani emulátor programokkal, azonban a kifogástalan működésért senki nem vállalja a felelősséget. Hiba esetén nyilvánvaló, hogy a felelősséget arra fogják hárítani, aki a "nem megfelelő" környezetre ültette át a programot, függetlenül a hiba tényleges okától.

Több ilyen – szintén kötelező – szoftver már kiváltásra került, és az irány akár kedvező is lehetne, hiszen ezek többsége böngészőben fut, mégis az ilyen programok közül alig kettőről mondható el, hogy bármilyen operációs rendszeren, bármilyen böngészőben használható. Ennek oka, hogy a többségük vagy böngészőfüggő (például az alkalmazás által generált URL-ekben backslash található) vagy olyan bővítményt igényel, amelyet nem minden böngésző támogat. Nem csak az operációs rendszer kiváltását nehezítik azonban azok a központi szoftverek, amelyek használata eltérő verziószámú bővítményeket igényelnek, hiszen ezek egyazon számítógépen történő futtatása még a nem nyílt forráskódú rendszerek használata esetén is komoly fejtörést okoz.

A közigazgatási rendszerek egyik legfontosabb eleme az iktató program. A 24/2006 BM-IHM-NKÖM egységes rendelet alapján ezeket a szoftvereket tanúsíttatni kell. Nyilvánvaló, hogy olyan ingyenesen hozzáférhető iratkezelő rendszer megjelenése nem várható a piacon, amely a jogszabály szerinti mindenkori tanúsítással bír, hiszen a tanúsításnak folyamatosan jelentős financiális vonzata van. Megoldást jelent egy platformfüggetlen módon futó szoftver, mivel így legalább a járulékos költségek (amelyek jellemzően a kliens és szerver oldali szoftverlicencek) ára megtakaríthatóvá válik.

3.4. Nem támogatott hardver

Gondot jelenthetnek a kliens oldali operációs rendszerek cseréjében a régi, nem támogatott hardvereket (nyomtatókat, szkennereket, fényképezőgépeket stb.) működésre bírni. Az átállás ebben az esetben erősen függ a későbbiek során megvásárolandó platformfüggetlen működéssel bíró eszközök beszerzésének időpontjától.

4. Megoldások

Ahogy a bevezetőben megfogalmazásra került, a közigazgatási folyamatok támogatását nyújtó szoftverekkel szemben támasztott elvárások jellemzően nem túl magasak. Általában szövegszerkesztő programra, számolótáblákra, elektronikus levelezőrendszerre, internet böngésző programra van szükség. Drága tervező és szerkesztő programokat nem, vagy csak korlátozott számban használnak. A speciális igényeket kielégítő backoffice szoftverek nagy többségükben tipizálhatók. Ha a felsorolt szoftvereket sikerül platformfüggetlen módon futtatni, a kliens oldali operációs rendszerek és irodai alkalmazáscsomagok gond nélkül cserélhetők nyílt forráskódúra, amivel a szegedi tapasztalatok szerint több száz millió forint megtakarítás érhető el egy-egy nagyobb hivatalban.

Szerver oldali megoldásoknál eddig is jóval több nyílt forráskódú szoftverrel találkozhattunk, gyakorlatilag minden informatikai problémára létezik megbízható, jól működő nyílt forráskódú alternatíva.

A legkézenfekvőbb megoldásnak az alkalmazásszolgáltató központ(ok) (ASP) létrehozása tűnik, ahol központilag futtatott, vékonykliens vagy böngészőben futó platformfüggetlen megoldásokat kapnának a közigazgatási szervek. Így – többek között – a teljes magyar közigazgatás kliens oldali szoftver ráfordításai megtakaríthatók lennének.

Szeged Megyei Jogú Város Önkormányzata rendelkezik olyan, saját tulajdonban lévő, alkalmazásszolgáltatásra is alkalmas integrált önkormányzati rendszerrel, amely az önkormányzatok feladatainak nagy részét támogatja. Szerver oldalon csak nyílt forráskódú megoldásokat igényel, kliens oldalon pedig platformfüggetlen módon böngészőben fut.

Szeged M. J. Város Közgyűlése a 454/2010. (VIII.13.) Kgy. sz. határozatában országos ASP központ létrehozása esetére felajánlotta az e-önkormányzati rendszerét a Magyar Köztársaság Kormánya és önkormányzatai számára ingyenes felhasználásra.

LibreOffice

Tímár András

Tartalomjegyzék

1. A kezdetektől a nyílt forrásig **138**

2. Az OpenOffice.org felemelkedése és bukása **138**

3. A LibreOffice elindulása **140**

4. Középpontban a fejlesztők **141**

5. A Document Foundation **142**

6. Az üzleti modell **142**

1. A kezdetektől a nyílt forrásig

A LibreOffice projekt 2010. szeptember 28-án indult el, de köztudott, hogy nem a semmiből tűnt elő. A programcsomag történetét 1985-ig vezethetjük vissza, ebben az évben alapította meg az ekkor 16 éves Mark Börries a Star Division nevű szoftverfejlesztő céget Németországban. A Star Division 1986-ban kezdte el fejleszteni a StarWriter szövegszerkesztőt az akkori legelterjedtebb, PC-n futó operációs rendszerre, DOS-ra. Az 1990-es évek elején megszületett a termék windowsos, majd OS/2-es és Mac-es verziója, a StarWriter mellé elkészítették a StarCalc táblázatkezelőt, a StarChart grafikonkészítőt, a StarDraw vektoros rajzolóprogramot és a StarImage bitképes rajzolóprogramot, és az egész csomagot StarOffice néven hozták forgalomba. A StarOffice 3.1-es verziója 1996-ban jelent meg, ez a változat már egy böngészőt és egy HTML-szerkesztőt is tartalmazott. A StarOffice 1997-ben vált teljes irodai csomaggá, amikor már bemutatókészítőt (StarImpress) és adatbázis-kezelőt (StarBase) is tartalmazott.

A Star Division önálló élete 1999-ben ért véget, amikor a Sun Microsystems 73,5 millió dollárért felvásárolta a céget. Simon Phippstől – aki akkoriban a Sun alkalmazottja volt – származik egy anekdota a felvásárlás okáról. Akkoriban a Sunnak kb. 42 ezer alkalmazottja volt. Nagy részüknek volt unixos munkaállomása és windowsos laptopja is. Olcsóbb volt venni egy olyan céget, amely Linux és Solaris környezetben is működő terméket állított elő, mint vásárolni 42 ezer Microsoft Office licencet a Microsofttól. Vagy nagyon drága volt 1999-ben egy Microsoft Office licenc, vagy Simon Phipps kicsit kiszínezte a történetet, esetleg nem is ezt mondta, de tény az, hogy az üzlet létrejött. A Sun a Microsoft Office versenytársává akarta tenni a terméket. Első intézkedésként ingyenesen letölthetővé tették a StarOffice 5.2-es verzióját.

2. Az OpenOffice.org felemelkedése és bukása

Az ezredforduló táján a nyílt forrású, illetve szabad szoftverek már komoly sikereket értek el, az üzleti élet is felfigyelt erre. A nyílt forrású fejlesztési modell szimpatikus volt néhány vállalatnak, ezek sorába lépett be a Sun is, amikor elkezdte megtisztítani a StarOffice kódbázisát a harmadik fél szerzői joga alatt álló elemektől, és végül 2000. október 13-án OpenOffice.org néven szabaddá tette az irodai csomag forráskódját LGPL/SISSL kettős licenc alatt. Az LGPL a Free Software Foundation által kiadott GNU Lesser General Public License, biztosítja a négy legfontosabb szabadságjogot, a kód szabad felhasználását tetszőleges célra, a kód tanulmányozásának és módosításának jogát, a szabad továbbterjesztés jogát, illetve a módosított verziók továbbadásának jogát. A SISSL a Sun Industry Standards Source License rövidítése, ezt a Sun fejlesztette ki. A SISSL egy gyenge copyleft típusú licenc, amely megengedi a származtatott munkák forráskódjának bezárását. 2005-ben az OpenOffice.org 2.0 már csak LGPL alatt jelent meg, a SISSL választhatósága megszűnt.

Eredetileg az OpenOffice nevet szerették volna adni a projektnek, de sajnos néhány országban ez a név már védett volt. Így lett az internetes tartománynév a projekt és a termék neve is. Körülbelül másfél évet vett igénybe, míg a szabaddá tett forráskódból egy vállalható terméket sikerült előállítani, ez kapta az 1.0-s verziószámot. A Star Division megszűnt, mint önálló cég, de megmaradt, mint a Sunon belüli önálló üzleti egység. Az OpenOffice.org fejlesztését továbbra is ez a társaság tartotta kézben.

Bár 2000-ben még olyan nyilatkozat látott napvilágot, hogy a Sun egy független alapítvány kezébe adná a forráskódot és a jogokat, a nyílt forráskódú fejlesztési modellt inkább úgy képzelték el, hogy az irányítást és a jogokat maguknak megtartva közösséget szerveznek a termék köré, és a saját fejlesztőkön kívül a külsős fejlesztők ingyenes munkáját is beépítik a termékbe. Bevételt az OpenOffice.org kereskedelmi verziójának, a StarOffice-nak a licencdíjából és a hozzá eladott terméktámogatási szerződésekből reméltek. Kiderült azonban, hogy egy nyílt forrású projekten dolgozó fej-

lesztőközösség létrehozásához nem elég a kódot megnyitni. Az első években külsős fejlesztőtől nem érkezett jelentősnek mondható kódhozzájárulás, a Sunon kívüli közösség szerepe inkább a honosításokban, a minőségbiztosításban és a dokumentációírásban domborodott ki. Az okok sokfélék. A Star Division Team vállalati kultúrájától idegen volt a nyílt forrású fejlesztés dinamizmusa, sokszínűsége. A forráskód módosításainak (patchek vagy magyarul foltok) befogadása extrém lassú volt. A külsős fejlesztők elé bürokratikus és technikai akadályokat állítottak, például: bonyolult specifikációs dokumentum kitöltése, elfogadtatása egy bizottság által, teszt build készítése legalább két platformra. A sikertelenséget látva próbáltak javítani a folyamatokon, némi sikerrel, de nemcsak ez volt a probléma. A szerzői jogok Sunnal való kötelező megosztása gondot okozott néhány egyéni hozzájárulónak is, de az OpenOffice.org-ban érdekelt, azonban a Sun versenytársának számító vállalatoknak végképp.

A fejlesztés így több ágra szakadt, és az egységes, közös munka helyett az egyes szállítók magukban dolgoztak. Egyesek megmaradtak a nyílt forrásnál. Például ilyen volt a Go-OO fork, amely kb. 800 saját foltot tett az OpenOffice.org forráskódjához. Szinte minden Linux-disztribúció ezt használta, és ezen alapult a Novell saját ügyfeleinek szállított windowsos buildje is. Mások kihasználták a SISSL által biztosított jogot, és nem adták ki saját kódjaikat (részleteket esetleg igen), például így tett az IBM a Symphonyval vagy a magyar Multiráció Kft. a MagyarOffice/EuroOffice programcsomaggal.

2010-ben az Oracle felvásárolta a Sunt, és így az OpenOffice.org forráskódja és a Star Division Team is a birtokába került. A StarOffice-t Oracle Open Office-ra nevezték át, és megnyugtatták ügyfeleiket és a közvéleményt, hogy a fejlesztés folytatódik. Viszont a Sun által 2000-ben megígért, független OpenOffice.org alapítványról és a fejlesztés „demokratizálásáról" továbbra sem akartak hallani, pontosabban az ezt firtató kérdésekre kitérő válaszokat adtak. A közösség egyes prominens tagjaiban ekkor megérett az elhatározás: lépni kell, nem várhatnak az Oracle-re. Elkezdték titokban kidolgozni a független OpenOffice.org alapítvány terveit. A 2010-es OpenOffice.org konferencia Budapesten volt, és szeptember 2-án este, míg az Oracle-csapat és a mit sem sejtő többi résztvevő a Dunán hajókázott, 16 ember egy étteremben véglegesítette az alapítvány irányelveit, és megválasztotta az első, ideiglenes vezetőségét. 2010. szeptember 28-án jelentették be: megalakult a Document Foundation, és az általa támogatott, OpenOffice.org-ra alapuló termék neve ezentúl LibreOffice, mivel az OpenOffice.org név birtokosa az Oracle.

Az Oracle a meghívás ellenére nem csatlakozott Document Foundationhöz. Folytatták az OpenOffice.org fejlesztését. 2011 áprilisában azonban beszüntették a kereskedelmi verzió támogatását, és mint később nyilvánvalóvá lett, megkezdődött a Star Division Team leépítése. Mostanra (2011. november) már senki nem dolgozik az Oracle-nél OpenOffice.org-on. A Star Division Team megszűnése mindenképp nagy veszteség. Bár hozzáállásukkal időnként bosszúságot okoztak, sok fejlesztő a kezdetek óta részt vett a projektben, pótolhatatlan tudásra és tapasztalatra téve szert. Szerencsére néhányukat felvette a Red Hat, ők LibreOffice-t fognak fejleszteni. Mások az IBM-nél folytatták pályafutásukat, talán az ő munkájukból is profitál majd a közösség.

Az Oracle az OpenOffice.org forráskódját sajnos nem a Document Foundationnek, hanem az Apache Foundationnek adta át, ezért sajnos az OpenOffice.org és a LibreOffice újraegyesülése elmaradt, és nem is lesz lehetséges az eltérő licencelés miatt. Kétséges azonban, hogy az Apache OpenOffice.org mennyire lehet életképes. Jelenleg több súlyos problémájuk van. A projektben részt vevő úgynevezett committerek között kevés a programozó, márpedig a fejlesztés programozók nélkül nem lehetséges. A teljes infrastruktúrát migrálniuk kell az Apache rendszere alá. Az Apache által választott verziókezelő, az svn, visszalépés az eddig használt mercurialhoz képest. Az Apache licenccel nem kompatibilis (pl. LGPL) licencű függőségeket ki kell váltaniuk, elég sok ilyen van. Például a Németh László által fejlesztett helyesírás-ellenőrző, a Hunspell is ilyen. Az Apache nem gazdája a projektnek, csak az infrastruktúrát és az ideológiát adja, a termék fejlesztése és a kiadása a fejlesztői közösség feladata. Azon múlik tehát a sikerük, hogy lesz-e elég fejlesztőjük. De miért akarna egy független fejlesztő az Apache OpenOffice.org-hoz csatlakozni, amikor a már bizonyított

LibreOffice projektben – egy év alatt 9 stabil kiadás – is könnyen megvalósíthatja elképzeléseit...

3. A LibreOffice elindulása

Kanyarodjunk vissza a 2010 szeptemberi eseményekhez, a LibreOffice elindulásához. A Document Foundation és a LibreOffice tulajdonképpen egy kísérlet. Kísérlet arra, hogy bebizonyosodjon, hogy a szabad szoftveres fejlesztési módszertanok egy ekkora kódbázis esetén is alkalmazhatók, és üzleti szempontból is sikeres termék állítható elő. A kérdés az, hogy vajon lehet-e egy olyan irodai programcsomagot készíteni, amelyre a fejlesztői büszkék lehetnek, és amely mögött egy valódi, nyílt közösség áll.

A Document Foundation fontosnak tartja, hogy a LibreOffice független legyen a gyártóktól, illetve szállítóktól. A LibreOffice fejlesztői és támogatói bázisa sokszínű. Fejlesztőket ad a Novell (15), a Red Hat (5), a Canonical (1), a Debian (1), és folytathatnánk még a sort sok kisebb, kevésbé ismert vállalattal, illetve a több száz önkéntessel. A Google a Summer of Code projekten keresztül támogatja, 2011-ben hét egyetemista dolgozott a LibreOffice-on. A forráskód git repositoryja és a fejlesztői levelezőlista a freedesktop.org-on van. Elvi támogatást ad az FSF és a vele sok kérdésben szemben álló OSI. Még a „Boycott Novell” is üdvözölte a LibreOffice-t a Novell jelentős részvétele ellenére!

Fontos változás a Sun/Oracle időkhöz képest, hogy a LibreOffice-hoz való hozzájárulás esetén nincs szerzőijog-megosztás. Az új hozzájárulók munkája MPL/LGPLv3+ licenc alatt kerülhet be, de a szerzői jogokkal minden hozzájáruló maga rendelkezik. Ez nehézzé, csaknem lehetetlenné teszi például a licenc megváltoztatását, adott esetben ez hátrány is lehet. Azonban a szerzői jog megosztása nélkül a hozzájárulók nagyobb biztonságban érezhetik hozzájárulásukat, nem lehet kisajátítani azt. Arról nem is beszélve, hogy a projekthez csatlakozás sokkal egyszerűbb, a folt beküldése és a licenc elfogadása ugyanabban az e-mailben megtörténhet, nem kell aláírni, faxolni, levelet küldeni, jóváhagyásra várni stb. Az eredmény kézzel fogható. A LibreOffice több egyéni, önkéntes kódhozzájárulót vonzott magához egy év alatt, mint az OpenOffice.org az első 10 évben.

Az új hozzájárulók akkor kezdenek el szívesen dolgozni egy projekten, ha könnyű bekapcsolódni, és hamar sikerélményhez jutnak. Az OpenOffice.org-tól megörökölt forráskód azonban legendásan ronda. Ezért a LibreOffice projekt indulásának első napján elkezdődött egy masszív kódtisztítási folyamat, amely máig tart, és még sokáig fog munkát adni, de megéri, mert a fejlesztők saját későbbi munkájukat könnyítik meg vele. Mellékhatásként a program gyorsabb lesz, és kevesebb memóriát foglal. A forráskód egy jó része régi, akár 20-nál is több éves. Organikusan növekedett, a refaktorálást hanyagolták. A kor miatt egy régebbi C++-nyelvállapot figyelhető meg, a standard könyvtár 20 évvel ezelőtti nemléte miatti saját fejlesztésű alaposztályokkal, egy feladatra akár több párhuzamosan létező osztállyal is. A fejlesztők kedvelt eszköze volt a copy&paste, aminek folyományaként előfordul, hogy egy hibajavítás nincs minden helyen átvezetve. A forráskódba szúrt megjegyzések jelentős része németül van, annak ellenére, hogy a kód már 11 éve nyitott. Sok kódrészlet felesleges, futás közben soha nem adódik rá a vezérlés, vagy túlbonyolított a rég nem támogatott platformokat (Alpha, OS/2, Windows 3.1, Windows 9x) kezelő kódok miatt. Ezek a kódtisztítási feladatok sokszor egyszerűek, csak egy kis szorgalmat igényelnek. Az ilyen és más belépő szintű feladatok gyűjteménye az Easy Hacks. Sok új fejlesztő kezdte ezzel, és ismerkedett meg ezen keresztül a kódbázissal. Érdemes rákeresni az interneten az „Easy Hacks LibreOffice” kulcsszavakra, talán e cikk olvasója is talál ott magának való feladatot...

4. Középpontban a fejlesztők

Egy szabad szoftvert fejlesztő projektben sokféle szerepe lehet a hozzájárulóknak. Nagyon hasznos munkát végeznek a tesztelők, a honosítók, a dokumentációírók, a designerek, a webmesterek, az infrastruktúrát üzemeltető rendszergazdák vagy akár a marketingesek. Mindazonáltal a szoftverfejlesztés lényegéből adódóan a legfontosabb tevékenység mégiscsak a kódírás. Új kód nélkül nincs fejlődés, és a többiek munkája is értelmetlenné válik. A LibreOffice fejlesztésén kb. 30 főállású fejlesztő dolgozik, de nyilvánvaló, hogy ez a létszám kevés a több millió kódsor karbantartására, a hibák javítására és új funkciók fejlesztésére. A Document Foundation nagyon nagy hangsúlyt fektet az új kódhozzájárulók megnyerésére és megtartására. Már említettük, hogy nem kell aláírni a szerzőijog-megosztási nyilatkozatot, ez jelentősen leszállította a belépési korlátot. További könnyebbség, és a bürokrácia visszavágása, hogy nem szükséges a folthoz hibajegyet nyitni a Bugzillában. Elegendő a foltot e-mailben elküldeni a fejlesztői levelezőlistára, nyilatkozni a licencről (MPL/LGPLv3+) és kész. A folt akár aznap bekerül a gitbe. A harmadik-negyedik jó folt után a hozzájárulónak felajánlják az írásjogot, innentől maga dolgozhat. Ezt sokan kockázatosnak ítélhetik, mert mi van, ha valami rosszat csinál egy új, ismeretlen hozzájáruló. A gyakorlatban azonban az előnyök felülmúlják az esetleges hátrányokat. A múlt tapasztalata az volt, hogy a szigorú ellenőrzés kontraproduktív. Meg kell bízni a hozzájárulókban, hinni kell, hogy tudják, hogy mit csinálnak. Az esetleges hibákat később javíthatók, a git logból egyértelműen visszakövethető minden változtatás.

Sok (kezdő) szoftverfejlesztő félénk, introvertált típus, nehezen szánja rá magát, hogy elküldje egy projektnek első munkáját, mert tart attól, hogy a tapasztalt fejlesztők leszólják. A LibreOffice-nál nagyon odafigyelnek arra, hogy ne riasszák el az új hozzájárulókat. Nem illik leszólni egy kezdő munkáját, az az ő „gyermeke¨, neki valószínűleg tetszik. A fanyalgás – jó, jó, de lehetne jobb – is azt eredményezi, hogy az illető nem küld többet semmit. Sokkal jobb rögtön megdicsérni: „Nagyszerű, hogy küldted ezt a foltot, máris betettük a gitbe, egy kicsit kellett még reszelni rajta, nézd meg a logban, de nagyon köszönjük, hogy segítettél jobbá tenni a LibreOffice-t. Mi lesz a következő téma, amivel foglalkozni szeretnél? Gyere fel az IRC-re is, stb.” A legtöbb ember fejlődőképes. Megéri az elején egy kis energiát fektetni a kedves fogadtatásba és az esetleges hiányosságok javításába, mert a további hozzájárulások valószínűleg egyre jobbak lesznek. Később a hozzájáruló maga is egy új hozzájáruló mentora lehet, és ezt a pozitív felfogást adja tovább.

A fejlesztői levelezőlistán és az IRC-n a pozitív hozzáállás azonban csak azokra vonatkozik, akik kódot küldenek vagy szeretnének küldeni. Vannak akik megírják a jó ötletüket, hogy mit kellene megvalósítani. Mások felháborodottan írnak, hogy addig számukra használhatatlan a LibreOffice, amíg ez meg az a hiba nincs javítva, és hogyhogy nincs, és hogy lehet így kiadni stb. A helyzet az, hogy ötletekkel „tele a padlás”. Rengeteg ötlete van a fejlesztőknek is. Ami kevés, az az erőforrás mindezen ötletek megvalósításához. Nem kellenek a kóddal alá nem támasztott új ötletek. Új fejlesztők kellenek. Ami a hibákat illeti, hetente átlagosan bejelentenek 100-at és javítanak 10-et. Látható, hogy nincs egyensúly. A hibajavításnál előnyt élveznek a fizetős ügyfelek hibái, legalábbis ami a fizetett fejlesztőket illeti. A kiadások időalapúak. Természetesen nem jelenik meg a kiadás, ha fel sem telepíthető, vagy el sem indul, vagy nem lehet semmilyen dokumentumot betölteni, de nem kritikus hibával megjelenik, és az ismert hibák a kiadási megjegyzésben fel lesznek sorolva. Nyilván minden kiadás tartalmaz ismeretlen hibákat is. Ezek akkor jönnek elő, amikor a felhasználók használni kezdik a programot. Az a típusú fenyegetőzés, hogy „javítsátok a hibámat, mert különben nem használom – az amúgy ingyen közreadott – programot”, egyszerűen nevetséges. Volt olyan fejlesztő, aki félig viccesen azt mondta, hogy annyi hiba van már bejelentve, hogy nem kell több, a meglévőkkel is ellennénk a következő pár évben. Természetesen nem minden hiba egyformán súlyos. A reprodukálható programösszeomlásokat vagy a regressziókat mindenképp érdemes bejelenteni.

5. A Document Foundation

2010 szeptemberében megalakult a Document Foundation (TDF). Az alapítók az OpenOffice.org közösség vezetői közül kerültek ki, pontosabban azok közösségi vezetők határozták el az alapítvány létrehozását, akik nem voltak Oracle-alkalmazottak. A bejelentést követő hetekben az OpenOffice.org közösségből szinte mindenki, aki nem az Oracle-nél dolgozott, bejelentette hogy csatlakozni kíván a TDF-hez. A TDF alapítói lefektették az azóta is érvényes alapelveket, tanulva a múlt hibáiból, más sikeres szervezeteket megfigyelve és megpróbálva megalkotni a tökéletes szabad szoftveres szervezetet.

A TDF független. Ez nem azt jelenti, hogy tagja vagy vezetői között nincsenek jelen a LibreOfficeban érdekelt vállalatok alkalmazottai. Minden érdekelt fél jelen van, de a különböző vezető testületekben egyiknek sem lehet 1/3-nál nagyobb részesedése. A LibreOffice jövője nem egyetlen szponzor kezében van, a fejlesztőközösség sokszínű és kiegyensúlyozott.

A TDF meritokratikus. Ez azt jelenti, hogy annak van nagyobb szava, aki többet tesz le az asztalra. Ez logikus, hogy így van, összhangban van azzal az elvvel, hogy azok vezessék a projektet, akik fejlesztik. A TDF tagja az lehet, aki jelentős hozzájárulással segítette a LibreOffice-t. Ez lehet bármi, fejlesztés, honosítás, dokumentációírás, marketing stb. A tagság elnyeréséhez legalább 3 hónapos „múltat” kell igazolni, egyúttal vállalni kell további 6 hónap aktív közreműködést. Egy vagy két ajánlóra is szükség van a tagsági kérelem elbírálásakor. A tagoknak választójoguk van és választhatók is a TDF vezető testületeibe. 2011 októberében zajlott le az első választás, a TDF-et irányító 7 tagú (+3 póttag) igazgatótanácsot választotta meg a tagság. Az igazgatótanács feladata a közösség építése, irányítása, a hivatalos (hatósági) ügyek intézése, az adománygyűjtés, a pénzügyek kezelése és a stratégiaalkotás. Azonban LibreOffice fejlesztését illető kérdéseket eldöntő Engineering Steering Comittee tagjait nem a tagság választja, hanem a fejlesztők maguk közül jelölik ki, így nem fordulhat elő, hogy olyasmit szavaz meg a tagság, amit nem lehet megvalósítani vagy ellenkezik a projekten dolgozó fejlesztők elképzeléseivel. A tagnak jelentkezés folyamatos. Az elutasított embereknek ismét jelentkezhetnek, ha jobban alá tudják támasztani jelentkezésük jogosságát.

Jelenleg a TDF nem jogi személy, nincs bejegyezve sehol. A TDF képviseletét egy német nonprofit szervezet, az OpenOffice.org Deutschland e.V. látja el, ők fizetik a számlákat, ők fogadják a beérkező adományokat. A TDF alapításakor szándékosan nem rögzítették le a szervezeti formát, hogy akik csatlakozni szerettek volna, szabadon alakíthassák ki a közös véleményüket erről. Az előkészületek kicsit több időt vettek igénybe a vártnál, de a cél egy nagyon stabil és megbízható szervezet létrehozása volt, ehhez sok erőfeszítés volt szükséges. A TDF ideiglenes vezetősége több lehetőséget is áttekintett, és végül az a döntés született, hogy az alapítvány szervezeti formája a német jog szerinti „Stiftung” legyen. A TDF németországi bejegyzéséhez minimum 50 000 euró volt szükséges. Ezért elindítottak egy nyilvános adománygyűjtő kampányt, hogy az adományokból szedjék össze az 50 000 eurót. Körülbelül egy hónapot adtak a pénz összegyűjtésére. Sikertelenség esetén az Egyesült Királyságba vitték volna át a székhelyet, kicsit más jogi feltételek mellett. Az 50 000 euró azonban mindenki meglepetésére 8 nap alatt összejött, sőt az adományozók nem álltak le, körülbelül a kétszer ennyi gyűlt össze az adománygyűjtési időszak végére. Megkezdődtek a tárgyalások az egyes német szövetségi államokkal, hogy kiderüljön, hogy melyikben lenne a legjobb helyen az alapítvány. Jelenleg annyi publikus, hogy 3 szövetségi állam „van még versenyben”, 2011 végére valószínűleg minden eldől, és a TDF be lesz jegyezve.

6. Az üzleti modell

A LibreOffice fejlesztésében több profitorientált vállalat is érdekelt. Az üzleti modell a klasszikus nyílt forrású modell. A vállalatok által fizetett fejlesztők és az ingyen dolgozó önkéntesek együtt

dolgoznak a LibreOffice fejlesztésén. Az eredményekhez mindenki hozzájuthat az LGPL licenc feltételeinek megfelelően – akár ingyen. A LibreOffice-t fejlesztő vállalatok ügyfeleiknek 3. szintű (L3) terméktámogatást tudnak biztosítani, azaz képesek az ügyfélnél jelentkező hibákat a forráskód javításával megszüntetni. Az L3-as támogatás pénzbe kerül, általában a felhasználószámmal arányos éves előfizetési díjért vehető igénybe.

A TDF minden intézményi felhasználónak melegen ajánlja, hogy vegye igénybe a LibreOffice-t fejlesztő valamelyik vállalat által nyújtott L3-as terméktámogatási szolgáltatást. Ez kölcsönös előnyökkel jár. Először is, az előfizetési díj még mindig sokkal olcsóbb, mint a versenytárs irodai csomagok licencdíja. Másodszor, sok bosszúságtól kíméli meg magát a szervezet, ha kidolgozott módszertan alapján kerül bevezetésre a LibreOffice, nem ad hoc módon. Harmadszor, a bevezetés és a használat során felmerülő hibákat a fejlesztők kijavítják. Fontos hangsúlyozni, hogy az lehetetlen, hogy hibabejelentő rendszerbe bejelentett minden hibával – mind a 2500 nyitott hibajeggyel – foglalkozzanak. Az viszont nagyon is lehetséges, hogy egy adott szervezetnél jelentkező, a napi használatot akadályozó 15-20 hibát kijavítsák. Nincs olyan felhasználó, akit minden hiba érint. A pénzéért cserébe a fejlesztők az ő hibáját fogják kijavítani. Negyedszer, ha mindenki ingyen veszi igénybe a fejlesztőközösség munkáját, akkor nem lesz bevétel, nem lesznek fizetett fejlesztők, és a LibreOffice fejlesztése óhatatlanul lelassul. Hosszú távon ezzel mindenki veszítene.

A TDF a jövő évtől kezdve hivatalos partneri kapcsolatot épít ki minden vállalattal, akik képesek L3-as szintű terméktámogatást nyújtani. A potenciális ügyfelek rájuk találhatnak majd a TDF, illetve a LibreOffice honlapján keresztül. Ennek a „reklámozásnak” azonban az a feltétele, hogy az illető vállalat bizonyítsa, hogy képes az L3-as szintű terméktámogatásra. Ezt csak egyféleképpen lehet bizonyítani: javítófoltok beküldésével. A TDF azzal lép szövetségre, aki elfogadja a nyílt fejlesztési modellt, és nemcsak elvesz, hanem vissza is ad.

Magyarországon például a Novell nyújt L3 szintű terméktámogatást a LibreOffice-ra. A Novell magyarországi konzultációs részlege több ezer fős szervezeteknél végzett felmérési és bevezetési projektek során kialakult módszertannal felméri az aktuálisan használt irodai szoftverek használatát, javaslatot tesz és költségbecslést készít az átállásra, utána elvégzi a bevezetést fejlesztői támogatással, igény szerint forráskód módosítással. A hibajavításokat a 15 fős SUSE LibreOffice Team végzi, amely messze a legnagyobb LibreOffice-os fejlesztőcsapat a többi TDF-partner vállalatéhoz viszonyítva. Nemcsak a SUSE saját fejlesztésű Linuxán a SUSE Enterprise Desktopon, hanem Windowson is támogatott a LibreOffice, jelenleg a 3.4-es verzió.

Az Ubuntu és a Unity

Torma László

Tartalomjegyzék

1. Az Ubuntu **146**

2. A Unity **147**

3. A Unity felépítése és működése **147**

4. A Unity és a felhasználók **149**

5. A jövő **149**

6. Ne féljünk az újdonságoktól! **150**

1. Az Ubuntu

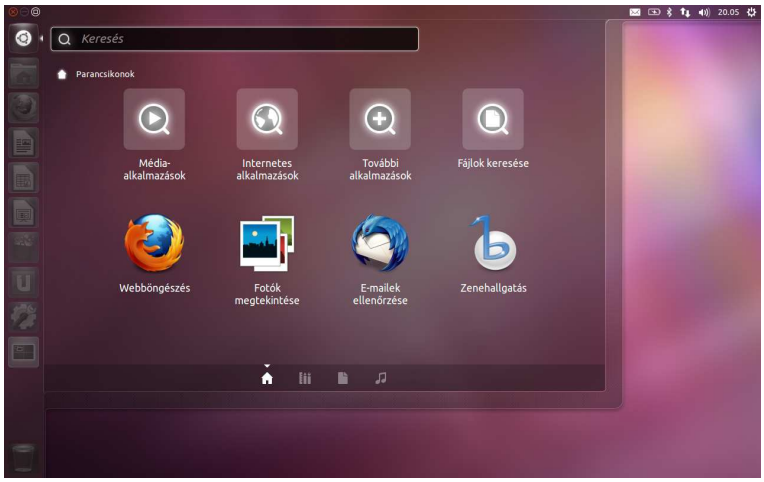
Hét év telt el azóta, hogy 2004 októberében megjelent egy új, Debian alapokra épülő GNU/Linux disztribúció, az Ubuntu. Az azóta eltelt időben a disztribúció óriási népszerűségre tett szert a felhasználók körében, és ugyan a Linux desktop éve továbbra sem köszöntött be, az Ubuntu mégis sikeresen bekerült a köztudatba Magyarországon is: az informatikai portálok rendszeresen foglalkoznak az Ubuntuval kapcsolatos fejleményekkel, és egy-egy új kiadás megjelenéséről a nagyobb híroldalak is beszámolnak. Az Ubuntu lassan a közszférában is kezd teret nyerni, egyre több hazai önkormányzatnál, közoktatási és felsőoktatási intézményben használják aktívan.

A rendszer óriási fejlődésen ment keresztül az első kiadás 2004-es megjelenése után: az elmúlt hét évben 15 Ubuntu kiadás jelent meg, melyek mind újabb és újabb fejlesztéseket, szolgáltatásokat vezettek be. Az Ubuntu által követett fejlesztési modell alapvetően két, egymásra épülő szakaszból áll: az egyes új kiadások fél éves ciklusokban (minden tavasszal és ősszel) követik egymást. Két évente pedig az úgynevezett LTS (Long Term Support), vagyis hosszú távon támogatott kiadások következnek, melyek egyben fontos mérföldköveket jelentenek a disztribúció életében. A fejlesztők célja, hogy a két évente megjelenő LTS kiadások idejére a normál kiadásokban megalapozott fejlesztések kiforrottak és jól működők legyenek. Ez már csak azért is kritikus, mert az Ubuntu következő, várhatóan 2012 áprilisában megjelenő LTS kiadása 5 éves támogatást kap kiszolgálókon és desktopokon egyaránt, így nem mindegy, mennyire nyújt majd ehhez stabil alapot a rendszer.

Az Ubuntu rendkívül gyors fejlesztési üteméből következik, hogy a rendszer hálás témát nyújt mindazoknak, akik fogékonyak az újdonságokra. Az Ubuntu a kezdetektől fogva nagy hangsúlyt fektetett arra, hogy az új fejlesztéseket azonnal elérhetővé tegyék az érdeklődők számára, amint azok valamilyen szinten működőképesek, vagy legalábbis elindulnak: az Ubuntu fejlesztés alatt álló kiadása már a fejlesztési ciklus legelejétől elérhető tesztelésre, így a bátrabb felhasználók folyamatosan nyomon követhetik, ahogy egy-egy kiadás formálódik a fejlesztési ciklus alatt. Természetesen ezek a fejlesztői kiadások nem alkalmasak éles használatra, de (akár napi szintű) tesztelésre tökéletesek, és nagyon nagy segítséget jelentenek azok számára, akik nem csak hírportálokon szeretnek olvasni az újdonságokról, hanem szeretik ki is próbálni azokat (és szívesen találkoznak elsőként a fejlesztői kiadásokkal járó bugokkal).

2. A Unity

Az Ubuntu történetének eddigi legjelentősebb változására 2011 áprilisában került sor: ekkor vezették be az új, Unity elnevezésű felhasználói felületet. A Unity egy új generációs felhasználói felület, melyet eleve úgy kezdtek el tervezni, hogy a későbbiekben a klasszikus, billentyűzettel és egérrel rendelkező számítógépek mellett érintőképernyős eszközökön is megállja a helyét. Ez nem azt jelenti, hogy a felületet kifejezetten táblagépekre szánták: a fejlesztés első szakaszában elsősorban a klasszikus gépekre (asztali munkaállomás, notebook és netbook) fókuszáltak, és a tervek szerint csak a 2012. áprilisi Ubuntu kiadás megjelenése után kezdenek bele az újabb eszközök (például táblagépek) magasabb szintű támogatásába.



A Unity első, kipróbálható verzióját 2010. május 10-én tették elérhetővé egy tárolóból, amit felvéve bárki telepíthette ezt a korai változatot a saját számítógépére. A Unity legelső verziója, bár működött, jóval kevesebb funkciót nyújtott, mint ma. A fejlesztők az azóta eltelt másfél év során rendkívül komoly fejlesztési ütemet diktáltak. A Unity az Ubuntu 10.10 Netbook Edition változatban vált először alapértelmezett felületté. Ekkor még a felület a GNOME Shell ablakkezelőjét, a Muttert használta, az Ubuntu 10.10 megjelenése után azonban teljesítményproblémák miatt portolták a felületet az Ubuntu által korábban is használt Compizra. A látványos effektusokat biztosító ablakkezelő az Ubuntu 7.10-es kiadásában lett alapértelmezett a grafikus gyorsítást támogató gépeken, így már volt ideje bizonyítani, hogy megfelelően gyors és stabil. Hogy a munka zavartalan legyen, a Canonical az egyik Compiz fejlesztőt, Sam Spillsburyt alkalmazta.

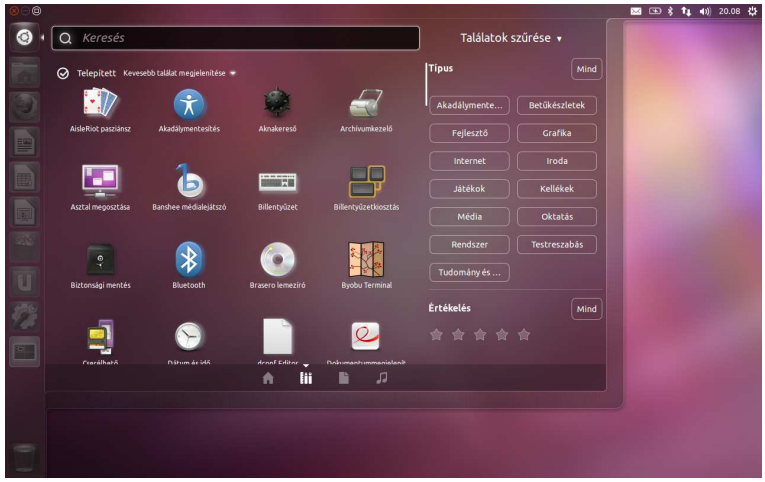
3. A Unity felépítése és működése

A Unity hardveres grafikus gyorsítást használó változata, a Unity 3D Compiz pluginként készült el. A hardveres támogatással nem rendelkező gépeken a Unity 2D indul el, amely a Qt eszközkészletet használja, és a QML elnevezésű, felhasználói felületek készítését nagyban megkönnyítő programozási nyelvre épül. A Unity 2D felületet első ránézésre szinte alig lehet megkülönböztetni a Unity 3D-től, egy-két effektustól eltekintve lényegében ugyanazt kapják a felhasználók. A Unity 2D, köszönhetően a barátságos erőforrásigényének, akár a grafikus gyorsítást támogató, de kisebb teljesítményű gépek felhasználói számára is jó választás lehet.

A Unity felület két főbb eleme a képernyő bal oldalán található indítópanel, valamint a képernyő tetején elhelyezkedő felső panel. A felhasználók az indítópanelen rögzíthetik leggyakrabban

használt alkalmazásaik ikonját, valamint ugyanitt jelennek meg az éppen futó alkalmazások is. A felső panelen érhető el az aktív alkalmazás menüje, a teljes képernyőre rakott ablakok ablakkezelő gombjai (bezárás, normál méret, minimalizálás), és a legfontosabb értesítések, köztük az üzenetek, a hálózati kapcsolatok, a hangerő, a naptár és a beállítások, valamint a kijelentkezésre és a rendszer leállítására szolgáló menü.

A bal oldali indítópanelen látható Ubuntu ikonra kattintva érhetjük el a Dash elnevezésű felületet, melynek segítségével alkalmazásokat indíthatunk el, kereshetünk korábbi dokumentumaink és a zenéink között. Ez a Scopes és Lenses elnevezésű megoldásra épül. A Scopes az egyes adatforrások (például alkalmazások, fájlok, zenék) összefogására és rendszerezésére szolgál, míg a Lenses az ezek megjelenítésére szolgáló felületet. A Scopes mögött álló technológia lehetővé teszi például az értékelés (*csak azokat az 5 csillagos alkalmazásokat szeretném látni,...*) és a kategóriák (*... amelyek a Játék kategóriába tartoznak*) szerinti szűrést. A megoldás mögött a Zeitgeist nevű motor működik.



A Zeitgeist mögött álló technológia a GNOME Activity Journal elnevezésű alkalmazással mutatkozott be. Sokan a mai napig az Activity Journallel azonosítják a Zeitgeistot, pedig valójában két különböző dologról van szó: a Zeitgeist egy eseményrögzítő motor, ami naplózza a különféle aktivitásainkat, míg az Activity Journal egyszerűen csak egy alkalmazás, amelynek segítségével megjeleníthetjük a naplózott eseményeket. A Zeitgeist nem fájlindexelő program, mint mondjuk a Tracker vagy a Beagle, csak a felhasználói aktivitást rögzíti.

A Zeitgeist daemon folyamatosan fut a háttérben, és az egyes alkalmazások ennek segítségével tárolhatnak el információkat az adatbázisban, erre szolgálnak a dataproviders gyűjtőnévvel ellátott pluginek. A Rhythmbox és a Banshee már upstreamben tartalmazza a Zeitgeist támogatást, de léteznek kiegészítők más alkalmazásokhoz is. A zeitgeist-datahub nevű Zeitgeist kiegészítő pedig azért felel, hogy a GtkRecentManager (Legutóbbi dokumentumok) adatai is bekerüljenek a Zeitgeist adatbázisába. A Zeitgeist eredetileg Pythonben íródott, azonban a projekt fejlesztői 2011. november 3-án bejelentették, hogy az alkalmazást a 0.9- es kiadási ciklusban portolták Vala programozási nyelvre. Az új verzió API szinten kompatibilis a régivel, így nincs szükség a már meglévő alkalmazások módosítására. A portolás eredményeként jelentősen csökkent az alkalmazás elindulási ideje és erőforrásigénye is.

4. A Unity és a felhasználók

Az Ubuntu mögött álló vállalat, a Canonical több tesztet is végzett ellenőrzött körülmények között laikus felhasználókkal, és az így szerzett tapasztalatokat és visszajelzéseket is felhasználták a felület kialakítása során. Az első tesztelésre 2010 októberében került sor, a Unity Ubuntu 10.10-be került változata alapján, majd 2011 áprilisában megismételték a tesztet, immáron a Unity legújabb, Compizra épülő változatával, a harmadik tesztelésre pedig az Ubuntu 11.10 megjelenésekor, 2011 októberében került sor. Az így szerzett visszacsatolás fontos szerepet játszott a felület kialakításában.

Mint az a fentiekből kiderül, a Unity fejlesztők nagy hangsúlyt fektettek arra, hogy kiszolgálják azon felhasználók igényeit, akik korábban nem használtak GNU/ Linux desktopot, és vonzó alternatívát kínáljanak számukra. Az ígéretek szerint a következő fejlesztési ciklusban a tapasztalt, profi felhasználók (power userek) igényei is nagy figyelmet kapnak . A cél az, hogy a tervek szerint 2012 áprilisában megjelenő Ubuntu 12.04 LTS már az ő elvárásaiknak is maximálisan meg tudjon felelni. Ez azért különösen fontos, mert az Ubuntu jelenlegi aktív felhasználói bázisának igen jelentős részét alkotják ezek a felhasználók. A másik fontos cél az Ubuntu 12.04 LTS kiadásában a stabilitás és a teljesítmény további javítása. Emellett nagy hangsúlyt fektetnek majd az akadálymentesítésre és a több monitoros rendszerek támogatására is.

5. A jövő

Az Ubuntu 12.04 LTS megjelenését követően újabb terület meghódítására készül az Ubuntu. Mark Shuttleworth, az Ubuntu alapítója az Ubuntu fejlesztők 2011 novemberében megrendezett konferenciáján (Ubuntu Developer Summit) bejelentette, hogy a disztribúció két éven belül a klasszikus számítógépek mellett okostelefonokon, táblagépeken és televíziókon is elérhető lesz. Az Ubuntut futtató okostelefonokra még várnunk kell egy kicsit: a fejlesztők az Ubuntu 12.04 LTS megjelenéséig arra fognak összpontosítani, hogy a rendszer a lehető legstabilabb legyen, és a Unity felületet tovább finomítsák, hogy egyaránt megfeleljen az otthoni felhasználók és a vállalati szféra igényeinek. Ezt követően helyeződik át a fókusz az újabb eszközök támogatására.

Mark Shuttleworth az UDS nyitóelőadásában kiemelte, hogy a világ átalakulóban van: egyre több intelligens eszköz jelenik meg, amelyek alkalmasak komoly, teljes értékű operációs rendszer futtatására, és képesek egymással és a clouddal kommunikálni. A Unity felület kifejlesztésével az Ubuntu egyik célja az volt, hogy a felhasználók számára egységes élményt nyújthasson a legkülönbözőbb eszközökön. A tervek szerint a Unity lehetővé teszi majd, hogy ugyanazt a felületet használjuk a telefonunkon, a táblagépünkön, az autónkban, a tévénken és persze a számítógépünkön. A Unity rugalmasan alkalmazkodni fog az egyes eszközök adottságaihoz, és bárhol megállja majd a helyét. Ennek gyakorlati megvalósítása komoly feladatot jelent, hiszen nyilvánvalóan teljesen másképp használunk egy asztali számítógépet, mint egy, a zsebünkben is kényelmesen elférő okostelefont vagy egy nagyképernyős televíziót.

A Unity felületet eleve úgy tervezték meg, hogy a későbbiekben támogassa az érintőképernyős eszközöket is. A multi-touch, vagyis a több ujjas érintés támogatása az Ubuntu 10.10 Netbook Edition megjelenésekor már része volt a rendszernek, tehát az alapok már most is megtalálhatók a Unityben. Ahhoz azonban, hogy a rendszer jól működjön az olyan eszközökön, mint a táblagépek vagy az okostelefonok, nagyon sok tennivaló van a felület kialakítása tekintetében. Ez nem csak magát a Unityt érinti, hanem az alkalmazásokat is, amelyeket úgy kell módosítani, hogy azokon az eszközökön is használhatók legyenek, amelyek nem rendelkeznek egérrel és billentyűzettel. Az egyik legnyilvánvalóbb példaként említhetőek a menük, amelyek remekül használhatók egérrel, érintőképernyőn azonban nagyon kényelmetlen lenne a használatuk. Vagyis nem elég magát a Unityt továbbfejleszteni, az alapvető alkalmazások felületét is alkalmassá kell tenni a táblagépeken és okostelefonokon való használathoz.

Nem csak a felszínen jól látható területeken van szükség fejlesztésre. Míg a klasszikus számítógépek tipikusan az x86 architektúrára épülnek, addig az új eszközök esetében (táblagépek, okostelefonok, okostévék) az alacsony fogyasztású, de egyre nagyobb teljesítményű ARM architektúra a meghatározó. Ezért különösen fontos, hogy az Ubuntu magas szinten támogassa ezt az architektúrát is. Ebben kulcsszerepe van a Linaro projektnek, melynek célja, hogy a nyílt forrású ARM fejlesztések számára hátteret és eszközöket biztosítsanak. A Linaro fejlesztők szorosan együttműködnek az Ubuntu fejlesztőkkel, ezt jelzi az is, hogy a Linaro fejlesztők találkozója, a Linaro Connect egy időben és egy helyszínen van az Ubuntu fejlesztők találkozójával, így az érdeklődők részt vehetnek egymás megbeszélésein, ezzel is segítve a két fél együttműködését.

A Canonical a Linaro mellett hardvergyártókkal is együttműködik, hogy az Ubuntu megjelenhessen a különböző mobil eszközökön. Mark Shuttleworth azt ígérte egy, a ZDNet által készített interjúban, hogy a fejlesztések eredményeit nyílttá teszik, és azok folyamatosan bekerülnek a standard Ubuntuba. A tervek szerint az Ubuntu 2014-ben megjelenő LTS kiadása már eszközök széles skáláját fogja támogatni. Ezt követően várható a különböző Ubuntus mobil készülékek széles körű megjelenése is.

6. Ne féljünk az újdonságoktól!

A 2011-es Szabad Szoftver Konferencia kiadványának olvasói között vélhetően jelentős számban vannak azok, akik nem csak használni akarják a számítógépüket, hanem szívesen elmélyednek a technikai érdekességek és új fejlesztések világában is. Azok, akik nem csak olvasni szeretnének a fejlesztés alatt álló Ubuntu kiadás újdonságairól, hanem maguk is szívesen kipróbálnák, a http://cdimages.ubuntu.com/daily-live/current/ címről közvetlenül letölthetik a fejlesztői változat legfrissebb napi buildjét. Ez, mint minden fejlesztői kiadás, természetesen nem alkalmas éles felhasználásra, hiszen nem stabil rendszerről van szó. Ugyanakkor arra tökéletesen megfelelő, hogy akár egy virtuális gépben futtatva, akár egy pendrive-ról elindítva teszteljük, hol tart éppen az Ubuntu fejlesztés alatt álló kiadása.

Amennyiben a tesztelés során esetleg belefutnánk valamilyen hibába, azt a Canonical fejlesztői portálján, a Launchpadon (https://launchpad.net) jelenthetjük. A hibajelentéssel kapcsolatos információk a https://wiki.ubuntu. com/Bugs/ és a https://help.launchpad.net/Bugs oldalakon olvashatók. Fontos, hogy mielőtt bármilyen bugot jelentenénk, győződjünk meg arról, korábban nem jelentette-e már azt valaki. Ha igen, akkor nézzük meg, hogy ki tudjuk-e egészíteni azt további információkkal, ezzel segítve a probléma gyors megoldását. Ha esetleg bizonytalanok vagyunk, hogy milyen információkat kell beleírnunk a bugreportba, bátran kérjünk segítséget IRC-n, a Freenode hálózat #ubuntu-bugs csatornáján vagy az adott projekt saját IRC csatornáján.

Az Ubuntu óriási utat tett meg azóta, hogy 2004 őszén megjelent az első kiadása. Az igazán izgalmas szakasz azonban még előttünk áll: ha minden a tervek szerint alakul, a kis, Debian alapú GNU/Linux disztribúció eljuthat oda, hogy asztali számítógépeken, notebookokon, netbookokon, táblagépeken, okostelefonokon, okostévéken, kis vállalati szervereken és gigantikus cloud kiszolgálókon egyaránt megállja a helyét. Az előttünk álló két év különösen izgalmasnak ígérkezik, érdemes lesz tehát folyamatosan nyomon követni a rendszer fejlődésével kapcsolatos híreket, és időről időre kipróbálni az éppen aktuális fejlesztői változatot.

RTF támogatás a LibreOffice Writer programban

Vajna Miklós <vmiklos@frugalware.org>

Kivonat

A LibreOffice RTF import és export támogatása igen elavult volt és a karbantartása nehézkessé vált. A cikkben bemutatom az új RTF importert (tokenizálót) és exportert, mely nem csak a későbbi fejlesztést teszi könnyebbé a gondos tervezésnek köszönhetően, de már most is számos többlet-támogatást nyújt az eredeti RTF filterekhez képest. A cikkben bemutatott új funkciókat a LibreOffice 3.5 már tartalmazni fogja minden RTF fájl olvasásához és írásához.

Tartalomjegyzék

1. A Google Summer of Code-ról **154**

2. A LibreOffice fejlesztésről **154**

3. RTF export fejlesztés **156**

4. RTF import fejlesztés **157**

5. Elérhetőségek **159**

1. A Google Summer of Code-ról

A GSoC egy pályázati lehetőség szervezetek és tanulók számára. Először szervezetek pályázhatnak ötletekkel, majd mikor az elfogadott szervezetek listája publikussá válik, tanulók jelentkezhetnek a szervezeteknél. Jelentkezéskor általában egy – a szervezet által kiírt – ötlet alapján kidolgozott pályázattal kopognak be a tanulók, de a kreatívabbak teljesen saját ötlettel is előállhatnak. Mikor lezárult a jelentkezési határidő, a pályázatok elbírálásra kerülnek, végül publikussá válik a sikeresen beválasztott tanulók listája.

A program móttója – *Flip Bits not Burgers* – arra utal, hogy eredetileg a cég ezzel azokat a tehetséges fiatalokat kívánta megcélozni, akik egyébként gyorséttermekben töltötték volna a nyarat, hogy pénzhez jussanak. A kezdeményezés lehetőséget biztosít arra, hogy szabad szoftver fejlesztéssel jussanak pénzhez.

A GSoC két ok miatt kerül említésre az előadáson:

- a bemutatott RTF filterek is hasonló finanszírozásban készültek el;
- reklámként is szolgál, hogy jövőre is legyenek olyan tanulók, akik a LibreOffice fejlesztésével szeretnék tölteni nyarukat.

2. A LibreOffice fejlesztésről

A LibreOffice projekt 2010. szeptember 28-án indult, alapítványi hátterét a The Document Foundation (röviden TDF) biztosítja. A szabad szoftveres projekt az OpenOffice.org forkjaként látta meg a napvilágot, a korábban patchset formájában létező Go-OO folytatásaként.

Fontos különbség, hogy míg a Go-OO a funkciók nagy részét hosszútávon vissza akarta juttatni az OpenOffice.org-ba, addig a LibreOffice esetén ezt feladták, így a projekt már a saját útján jár.

Ettől eltekintve persze a nagy kódbázis jelentős része megegyezik, így az OpenOffice.org projektben szerzett tapasztalatok itt is könnyen kamatoztathatóak.

Gyakori félreértés, hogy az emberek úgy gondolják: a LibreOffice Java nyelven íródott, holott ez nincs így. A kód nagy része C++, a maradék többségét valóban a Java teszi ki, de emellett még számos egyéb nyelven írt kisebb kódokat is tartalmaz a projekt (XML, Make, ASM, Yacc, Perl, Python stb.).

Kedvcsinálóként az előadáson ismertetésre kerülnek azok az első lépések, melyeket meg kell tennie azon önkénteseknek, akik szeretnének a projekthez programkóddal hozzájárulni.

Első build

Az első fordítás három lépésből áll:

1. forrás letöltése:

```
git clone git://anongit.freedesktop.org/libreoffice/core
```

2. fordítás

```
./autogen.sh
make
make dev-install
```

3. futtatás

```
cd install/program
source ./ooenv
./soffice.bin
```

Inkrementális build

Mivel egy teljes fordítás sok időt vesz igénybe, a következő kérdés, hogy a forráskódbeli módosítástól az új futtatható programig hogyan juthatunk el. Ennek módja, hogy a forráskód számos (jelenleg 226) modulra van osztva, melyek külön-külön is fordíthatóak.

Például ha a `writerfilter` modult módosítottuk, akkor lépjünk annak könyvtárába és futtassuk ott a `make` programot, aminek határására futtatáskor már életbe lépnek a változtatásaink.

A gyakorlatban néhány paramétert praktikus alkalmazni:

- `-s`: kevesebb kimenet kérése, hogy a hibákat, figyelmeztetéseket könnyebben észrevegyük;
- `-r`: a beépített implicit szabályok elhagyása, mely gyorsabb működést eredményez;
- `-j4`: többszálú fordítás (a 4 az aktuális gépen elérhető CPU-k vagy magok számával helyettesítendő);
- `dbglevel=2`: a fejlesztést segítő, `OSL_TRACE()` kimenetek bekapcsolása;
- `build`: csak fordítás, unit tesztek futtatásának elhagyása.

A teljes parancs tehát:

```
make -sr -j4 dbglevel=2 build
```

Aminek segítségével tipikusan tíz másodperc alá szorítható a fordítási idő. Természetesen az egyes paramétereket mindenki ízlése szerint megválaszthatja.

A projekt mérete

A LibreOffice az egyik legnagyobb méretű szabad szoftveres projekt. Voltak arra törekvések, hogy a forráskódot húsznál is több különálló tárolóba szétválasszák, de jelenleg olyan szoros az ezek közötti függés, hogy ez kudarcot vallott: a fejlesztés során nem vált lehetségessé ténylegesen csak egy-egy alrendszer módosítása. Ennek következtében jelenleg a módosítások nagy része egyetlen (az ún. `core`) tárolóban történik.

A `git clone` végeztével kb. 8 millió kódsorhoz jutunk, a merevlemezünkön pedig kb. 4 GB helyet veszítünk. A teljes fordítás ideje eltérő lehet: Linuxon szélső értékként egyrészt a gyakori fordítás (pl. tinderbox) és `ccache` használat mellett elérhető kb. 15 perces időigényt lehet említeni, a másik véglet ha minden nyelv támogatását bekapcsoljuk, akkor még egy négymagos gépen is 3–4 óra fordítási idővel kell számoljunk.[1]

[1] Ha lassúsági rekordot szeretnénk elérni, próbálkozzunk a fordítással egy manapság oly divatos netbookon.

Kollaboráció

A LibreOffice fejlesztése főként a következő kommunikációs csatornákon keresztül folyik:

- forráskód-kezelés: git;
- napi kommunikáció: IRC, e-mail;
- stratégiai döntések: Technical Steering Call (TSC).

RTF filterek

Filtereknek nevezzük azokat a programmodulokat, melyek a dokumentummodell betöltését vagy elmentését végzik valamilyen – lehetőleg szabványosított – formába. A cikk szerzője a továbbiakban a LibreOffice Writer alkalmazás RTF import/export filtereinek elkészítése során szerzett tapasztalatait, illetve élményeit ismerteti.

Az RTF formátumot sokan jelentéktelen szabványnak tartják, pedig a maga nemében egyedülálló:

- Az első verzióját 1987-ben fogadták el, megelőzve az XML-t vagy az ODF-et;
- Az újabb verziói visszafele kompatibilisak;
- Időről időre frissül, a jelenlegi (1.9.1-es) szabvány támogatja a beágyazott táblázatokat, OLE objektumokat stb.;
- Előszeretettel használják sok helyen (pl. kormányzati űrlapok), ahol az ODF még túlságosan újdonságnak számít.

Természetesen megvan az a hátránya, hogy az RTF nem egy nyílt szabvány, hanem a Microsoft adja ki az újabb verziókat.

3. RTF export fejlesztés

2010 nyarán egy teljesen új RTF exporter készült el. Az ötlet az volt, hogy az RTF sok aspektusban hasonlít a doc, illetve docx formátumokra (mindhárom Microsoft találmány), és a doc/docx exportereknek már van egy közös alapja. A korábban különálló RTF exporter helyett tehát egy – erre a közös alapra építkező – új RTF exporter készült el.

A szokásos célok (ne okozzon regressziót a régi filterhez képest, legyen kisebb, nyújtson több funkcionalitást) közül csak a méret csökkentése nem sikerült, az új funkciók helyigénye miatt.

Ez az exporter először a Go-OO 3.3 első teszt-kiadásában volt elérhető, azóta része a LibreOffice-nak. A LibreOffice megszületése előtt része lett az OpenOffice.org 3.4 bétának is, amiből sajnos azóta se látott a világ stabil kiadást.[2] Ettől függetlenül az Oracle híresen szőrszálhasogató QA-én átjutott az exporter, és ennek során számos hasznos visszajelzést kaptam a cég mérnökeitől.

Az exportert leginkább azoknak érdemes kipróbálniuk akiknek problémájuk akadt a régi változattal, de az érdeklődők számára az előadáson példákat mutattam a következő új funkciókra:

- könyvjelzőkre mutató oldalszám-hivatkozások;
- karakter tulajdonságok (kiterjesztett térköz, alávágás);

[2] Aki kíváncsi a fejleményekre, látogasson el az *Apache OpenOffice.org (incubating)* projekt fejlesztői listájára és kövesse az eseményeket.

- OLE objektumok (pl. diagram);
- rajzok;
- űrlapok;
- sorszámozás;
- matematikai kifejezések, szerkesztést lehetővé tevő natív adattal;
- oldaltörések;
- oldalszámozási stílusok, oldalszámozás újraindítása;
- képek Wordpadben;
- post-it mező;
- hasábtörés;
- védett szakaszok;
- beágyazott táblázatok;
- javított tartalomjegyzék.

Az új funkciók hosszú listája ellenére az új filter sem hibamentes, visszajelzéseket a Freedesktop Bugzillájába[3] kéretik eljuttatni.

4. RTF import fejlesztés

Idén nyáron került sor az új RTF importer elkészítésére, mely az exporterhez hasonló ötleten alapszik: a docx importerhez készült framework újrahasznosítható lenne több tokenizáló elkészítéséhez az 1. ábrán látható módon.

Jó példa az így egy helyen megvalósított funkciókra a mezők értelmezése (pl. oldalszám, annotációk) vagy a Writer oldalstílusai és a Word speciális fejlécei között.

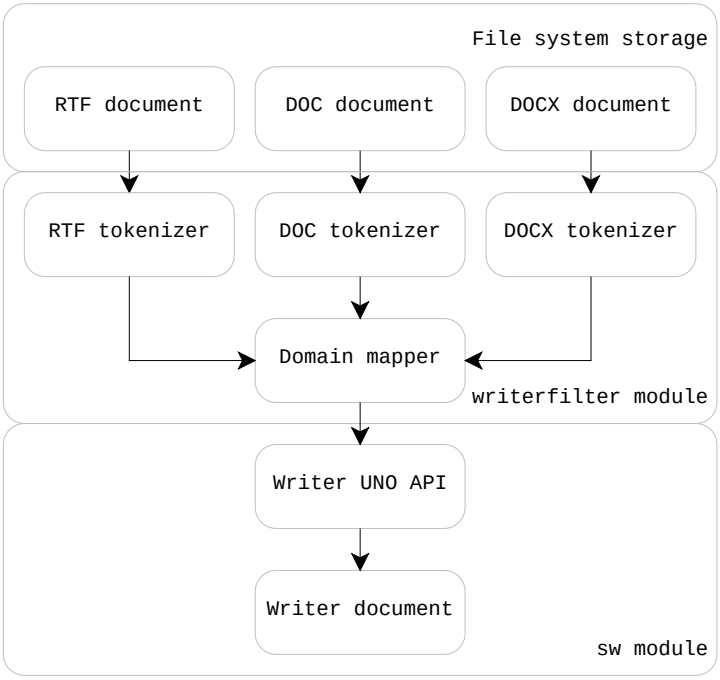
Tesztelés

A tesztelés első lépése a csak a tokenizert tesztelő unit teszt elkészítése volt. A minden fordítás során automatikusan lefutó ellenőrzés a korábbi RTF importer CVE hibáihoz tartozó teszt dokumentumokat értelmezi. Mivel csak a tokenizer kerül tesztelésre, a teljes futás kb. 200 ms.

A megoldás szépsége, hogy így már akkor lehetséges a teszteket végrehajtani, mikor a Writer alkalmazást megvalósító `sw` modul még nem került lefordításra.

Természetesen ez a teszt a kézi ellenőrzést nem helyettesíti, hiszen a vizuális megjelenést nem vizsgálja – csak azt, hogy a tokenizer elfogadja, vagy elutasítja az adott dokumentumot.

[3] http://bugs.freedesktop.org



1. ábra. Az RTF import filter architektúrája

Új funkciók

Az export filterhez hasonlóan az importer új funkcióiból is ízelítőt kapnak az előadás hallgatói. Bemutatásra kerülnek a korábban nem, vagy hiányosan, illetve rosszul támogatott következő elemek importálása:

- beágyazott táblázatok;
- lábjegyzetek;
- post-it mezők;
- űrlapok;
- rajzok;
- szövegdobozok.

Hatás a DOCX importra

A régi RTF importernek néhány funkcióját először azért nem lehetett megvalósítani az új filterben, mert a keretrendszer nem támogatta az adott funkciót. Ezek a módosítások a bekerülésük után a DOCX importert is javították, melyből néhány szintén demonstrálásra került az előadáson:

- dupla-áthúzás;
- lábjegyzetek újraszámozása.

5. Elérhetőségek

A szerző ezúton is elnézést kér, hogy sok – a cikkben szereplő – témát csak érintőlegesen említett, csak az import és export filterek témáról vastag könyvet lehetne írni, a cél leginkább a figyelemfelkeltés volt. Az alábbi linkek további kérdések esetén remélhetőleg segítséget nyújtanak.

- LibreOffice honlap: http://libreoffice.org/
- GSoC: http://code.google.com/soc/
- A diák és ezen cikk elérhetősége: http://vmiklos.hu/odp/

Többszálas (multi threaded) programozás Linux környezetben

Vomberg István <istvan@vomberg.hu>

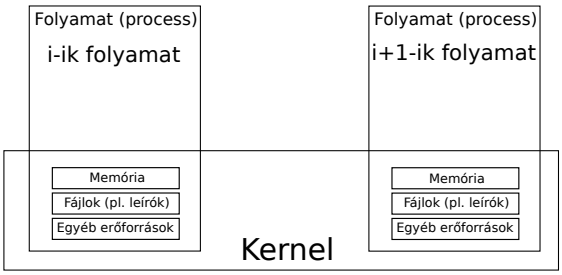
Tartalomjegyzék

1. Mik azok a szálak? 162
1.1. Mikor használunk szálakat? 162
1.2. LinuxThreads vs. Posix 163

2. A szálak kezelése 164
2.1. Nevezéktan 164
2.2. Szálak a gyakorlatban 164
2.3. Más rendszerek 166

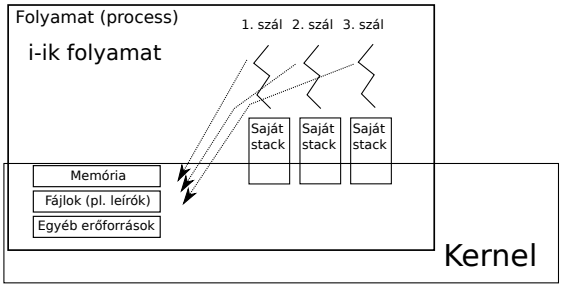
1. Mik azok a szálak?

A szálak hasonló egységek mint a folyamatok (processzek), ám néhány erőforrást nem közvetlenül a kerneltől kapnak hanem a meglévő folyamaton belül osztoznak rajtuk egymás közt. Magyarul a szálak azok egy folyamaton belüli alfolyamatok, melyek a meglévő folyamat erőforrásait használják megosztva.



1. ábra. A kernel és a folyamatok

A folyamaton belül a szálak a folyamat összes erőforrását közösen használják, egyedül a szálankénti saját stack az, amivel kizárólagosan gazdálkodhatnak. Természetesen ez azonnal mutat egy kézenfekvő problémát, a globális adatterületekhez minden szálnak szabad hozzáférése van, így egymás „feje fölött” átnyúlkálva keresztbe is módosíthatják azokat, amik reprodukálhatatlan összeomláshoz vezethetnek. Természetesen ennek a problémakörnek nagyon körbejárt feloldása van, ez később kerül ismertetésre.



2. ábra. A folyamat és a szálak

1.1. Mikor használunk szálakat?

Nyugodtan kijelenthetjük, hogy a programok több mint 90%-a egyáltalán nem igényel szálkezelést. Szálkezelésre az alábbi szempontok alapján tekinthetjük alanynak a kódot:

- Független a többi kódtól. Ezalatt például értsük azt, hogy a kód igényel-e adatokat más kódrészletektől illetve az eredményeitől függenek-e más kódrészletek.
- Hosszú ideig blokkolja a futást valamely művelete. Gyakori eset valamely lassú kommunikációs interface használata, pl. RS232.
- Hosszú futásidejű, a CPU-t terhelő műveletet végez-e? Ez szintén blokkolja pl. az IO műveleteket egy folyamaton (processzen) belül.

- Aszinkron műveleteket végez a kód, pl. hálózati kommunkikáció.
- Kevésbé fontos (alacsonyabb prioritású) műveletet végez a kód.

A felsorolásból látható, hogy a potenciális szálhasználó programok elsősorban a szerver programok, webszerver, printszerver, adatbázis kiszolgálók. Folyamatosan aszinkron eseményekre válaszolnak, majd IO műveleteket végeznek. Hasonló feladatok vannak a méréstechnikában, jellemzően több, lassabb perifériát, adatgyűjtőt kell kiszolgálni.

Amennyiben valóban multiprocesszoros gépre fejlesztünk és a rendszer szálkezelése a szálakat el is tudja osztani az egyes processzorok közt, úgy a jelfeldolgozás (pl. kép-, videofeldolgozás) is komolyan gyorsulhat a párhuzamosítható részek valódi párhuzamosításával. Alap példának a mátrix szorzást vagy a Mandelbrot-halmazok számítását szokták felhozni, ám tudni kell, hogy egyprocesszoros gépen a többszálas implementáció még lassulást is előidéz, mivel az egyes feladatok közti váltást adminisztrálni kell és a futás valójában nem párhuzamos, csak annak tűnik.

1.2. LinuxThreads vs. Posix

A Linuxban 1996-ban jelent meg a szálkezelés, formailag majdnem POSIX konform módon, azonban megvalósításában attól eltérően. A gondot az okozta, hogy a 2.4- es verzióig a kernel nem volt felkészítve a szálak kezelésére. A szálakat mint önálló kernel szálakat kezelte, s ez olyan következményekkel járt, amik a többszálú programok használatában okoztak sok gubancot. Mik voltak ezek a problémás pontok?

- Külön PID-je volt minden egyes szálnak.
- Be kellett vezetni egy *manager thread*-et, amely valójában kezelte az összes többit, pl. a létrhozásuk is ezen keresztül ment. Ez egyrészt szűk keresztmetszet volt, másrészt viszont ha ez a szál megszűnt (pl. `kill`), akkor a többit kézzel kellett egyenként kilőni. Ez több száz szál esetén komoly problémát jelentett.
- A jelek (`signal`) kezelése rendszeresen bedöntötte a programot, abszolút nem volt köze a jelkezelésnek a POSIX előírásokhoz.
- A jelkezelés problémáinak folyományaképp a `SIGSTOP` és a `SIGCONT` csak az adott szál futását állította le.
- Az IA-32 architektúrán 8192 szál volt a maximum. Ez nagyobb projekteknél (pl. webszerver) komoly korlátot jelentett.
- A rengeteg PID a `/proc` táblát használhatatlanná tette.

Ezen problémák nagyon hamar komoly korlátokat jelentettek a többszálú programozásban, így a munkák rövidesen megkezdődtek, melyek a POSIX kompatibilis `pthred` implementáció megírásához vezettek. A fontos szempontok a tervezésnél az alábbiak voltak:

- POSIX megfelelőség. Forráskód szinten kompatibilis legyen más POSIX rendszerekkel.
- A többprocesszoros rendszerek erőforrásainak tényleges kihasználása.
- „Olcsó”, azaz gyors és hatékony szál indítás.
- Azon programokat, melyek a pthred libraryval linkelve vannak, ez ne terhelje akkor sem, ha nem használnak szálakat.

- Bináris kompatibilitás a LinuxThreads implementációval.
- Jó skálázhatóság. Többprocesszoros rendszerekben ez legyen közel lineáris, az adminisztrációs igény komolyabb növekedése nélkül.
- A szoftveres limitek megszüntetése, pl. a szálak lehetséges száma.
- A nagygépes architekúrák támogatása. A nagygépes processzorok sokszor jelentősen különböznek a desktop gépektől, ezeken is megfelelően kell menniük a szálaknak.
- NUMA support, a non-uniform memory architectures támogatása.
- C++ integráció

Ezek a munkák 2002-ben tetőztek és eredményeképp létrejött a pthread library és a hozzá tartozó kernel támogatás is.

2. A szálak kezelése

A szálak kezelésével kapcsolatban három fő csoportba oszthatjuk az ezen feladatokhoz tartozó függvényeket:

- A szálak menedzselése: paraméterezés, létrehozás, csatolás.
- A mutexek: ezek használata biztosítja azt, hogy egy adatterülethez konkurens szálak úgy férjenek hozzá, hogy egymás tevékenységét ne üssék ellenőrizetlenül keresztbe. Ezek olyan szoftveres zárak, melyek biztosítják ezt, hogy egy adathoz egyszerre csak egy szál férhet hozzá, a többi addig várakozik.
- Feltételes változók: míg a mutexek az adat hozzáférése alapján működnek, a feltételes változók az adat értéke alapján teszik lehetővé egyes kódrészletek lefutását. Ez egyúttal a szálak közti kommunikációt is magában foglalja.

2.1. Nevezéktan

`pthread_`	A szálkezelő függvények nevének kezdete.
`pthread_attr_`	Szál attribútum függvények.
`pthread_mutex_`	Mutex függvények.
`pthread_mutexattr_`	Mutex attribútum függvények.
`pthread_cond_`	Feltételes változók.
`pthread_condattr_`	Feltételes változók attribútumai.
`pthread_key_`	Szálspecifikus kulcsok.

2.2. Szálak a gyakorlatban

Egy „csontváz” mintaprogram, melyen az alapelvek vázlata tanulmányozható. Van két függvényünk, a `do_1()` és a `do_2()`, melyek csinálnak valamit egy közös, globális változón (jelesül inkrementálják és kiírják az értékét), eközben biztosítják azt, hogy egymás feje fölött ne nyúlkáljanak át. Amennyiben futtatjuk eme programot, úgy egyesével növekvő számokat ír ki, melyekről látható lesz, hogy éppen melyik szálban történt a rajtuk végzett művelet.

```
#include <stdio.h>
#include <pthread.h>

void do_1(void);
void do_2(void);

int my_val = 0;
pthread_mutex_t my_mutex=PTHREAD_MUTEX_INITIALIZER;

int main (int argc, char **argv) {

  pthread_t thread1, thread2;
  pthread_attr_t custom_sched_attr;

  pthread_attr_init(&custom_sched_attr);
  pthread_attr_setscope(&custom_sched_attr, \
                        PTHREAD_SCOPE_SYSTEM);

  pthread_create(&thread1, &custom_sched_attr, \
                        (void *) do_1, NULL);
  pthread_create(&thread2, &custom_sched_attr, \
                        (void *) do_2, NULL);

  pthread_join(thread1,NULL);
  pthread_join(thread2,NULL);

  while (1) {};
  return 0;

}


void do_1()
{

  pthread_mutex_lock(&my_mutex);
  printf("Thread 1: %d\n", my_val++);
  pthread_mutex_unlock(&my_mutex);

}


void do_2()
{

  pthread_mutex_lock(&my_mutex);
  printf("Thread 2: %d\n", my_val++);
  pthread_mutex_unlock(&my_mutex);

}
```

2.3. Más rendszerek

A szálkezelés területén egy fix pontunk van amelyen keresztül be tudjuk sorolni a különféle rendszereket: POSIX alapú az adott implementáció vagy sem. A UNIX alapú rendszerek mindegyikéről elmondható, hogy a POSIX-hoz igazodnak, természetesen különbözőségek vannak, egyrészt hogy implementáltak-e mindent ami a POSIX szabványban van, illetve milyen plusz dolgokat nyújtanak még ezen felül. Forrás szinten jellemzően kompatibilisek – vagy minimális munkát igényelnek – ezeken a rendszereken a szálkezelést használó programok. Az Ablax világa egy teljesen más világ, a Win32 illetve az OS2 (majd NT) típusú szálkezelés alapjaiban különbözik az eddigiektől, a POSIX-ra leképezni sem lehet, ám ennek ellenére készül(t) POSIX megfelelőségű réteg ezen rendszerekre, erről mélyebb információim nincsenek, érzésem szerint a *„meg tudjuk csinálni ezt is, mert olyan jók vagyunk”* világmegváltási elv okán készülhetett el.